UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 6

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Herramienta para la generación de paquetes de instalación de personalizaciones de la plataforma GeneSIG.

AUTOR: Fidelky Alcántara Martínez

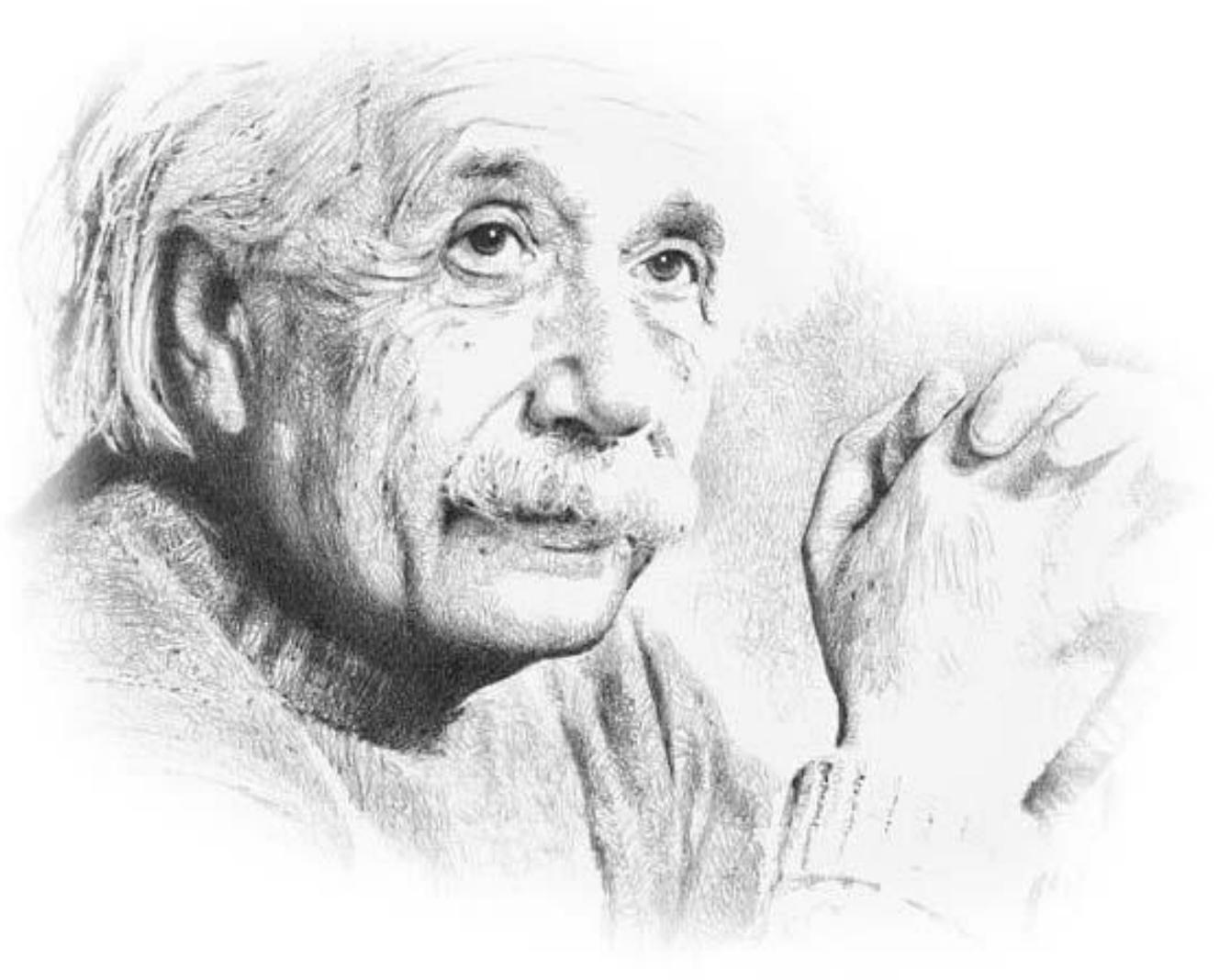
TUTOR: Ing. Rafael Enrique Naranjo Leonard

CO-TUTOR: Ing. Alain León Companioni

La Habana, junio del 2014

"Año 56 de la Revolución"

"La mente es como un paracaídas...
Solo funciona si la tenemos abierta"



Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Fidelky Alcántara Martínez

Autor

_____

Ing. Rafael Enrique Naranjo Leonard

Tutor

_____

Ing. Alain León Companioni

Co-Tutor

DATOS DE CONTACTO

Síntesis de los tutores:

Ing. Rafael Enrique Naranjo Leonard: graduado de Ingeniería en Ciencias Informáticas en la Universidad de las Ciencias Informáticas en el año 2013. Actualmente se desempeña como desarrollador de lógica de negocios de la Línea de Productos de Software Aplicativos SIG del centro Geoinformática y Señales Digitales.

Correo Electrónico: rnleonard@uci.cu

Ing. Alain León Companioni: graduado de Ingeniería en Ciencias Informáticas en la Universidad de las Ciencias Informáticas en el año 2010. Actualmente se desempeña como jefe de la Línea de Productos de Software Aplicativos SIG del centro Geoinformática y Señales Digitales.

Correo Electrónico: acompanioni@uci.cu

AGRADECIMIENTOS

Agradezco a mi madre Loyda, mi guía y principal inspiración, por hacer de mí un hombre de bien. Todo lo que soy es gracias a ti. Mita, gracias por existir.

A mis dos padres Jose y Fidel por ser ejemplos en mi vida, por darme su apoyo en todo momento a pesar de no siempre estar presentes físicamente, por inculcarme valores y por el sacrificio que pusieron en mí. Algún día espero ser como ustedes.

A mi esposa y amiga Zahily por estar siempre a mi lado durante todos estos años y por ser una de las cosas más importantes que me han pasado en la vida.

A mis abuelos, especialmente a mi abuelo Juancito por su cariño desmedido y por las tantas veces que lo hice cantarme las mismas décimas y canciones que me componía cuando era pequeño. Gracias abuelo, nunca las olvidaré.

A mis tutores y amigos Rafael y Alain por guiarme en la realización de este trabajo.

A mis compañeros de grupo por compartir estos cinco años de experiencias y amistad.

A mis amigos de la radio por hacer de este hobby otra familia.

A todos, gracias por formar parte de este sueño.

DEDICATORIA

A mis padres por su amor y apoyo incondicional.

A mi esposa por estar siempre a mi lado.

A mis abuelos por su cariño.

A mi familia.

RESUMEN

En el centro de desarrollo de software Geoinformática y Señales Digitales se desarrollan una amplia gama de productos que son empleados en diversas esferas de la sociedad en Cuba. Dentro de las soluciones realizadas por este centro se encuentran los Sistemas de Información Geográfica construidos a partir de la plataforma soberana GeneSIG. El despliegue de las personalizaciones de GeneSIG se realiza de forma manual lo que provoca que el proceso se torne lento y complejo cometiéndose errores en las configuraciones o dejando dependencias sin instalar. Ofrecer solución a estos errores ocupa gran cantidad de tiempo y esfuerzo lo que provoca en ocasiones el reinicio del proceso de instalación. Para dar respuesta a la problemática se desarrolló una herramienta que permite generar paquetes de instalación de personalizaciones de la plataforma soberana GeneSIG.

El desarrollo de la herramienta estuvo guiado por el Proceso Unificado Abierto haciendo uso del Visual Paradigm para el modelado, Java como lenguaje de programación y Netbeans como entorno de desarrollo integrado. La herramienta obtenida permite la creación de instaladores con un elevado nivel de configuración y detalle. Estos, guían la instalación a través de un asistente detallado donde no es esencial que el usuario posea conocimientos avanzados de informática.

Palabras claves: GeneSIG, paquete de instalación, Sistema de Información Geográfica.

ABSTRACT

In the center of software development Geoinformatics and Digital Signals develop a wide range of products that are used in various areas of society in Cuba. Among the solutions provided by this center are the Geographic Information Systems constructed from the GeneSIG sovereign platform. The deployment of GeneSIG customizations is done manually which causes the process becomes slow and complex causing errors in the configurations or neglecting dependencies installation. Provide solution to these errors takes up lots of time and effort which causes sometimes restart the installation process. To solve the problem development is a tool that permit generate installation packages of customizations GeneSIG platform.

The development of the tool was guided by the Open Unified Process using Visual Paradigm for modeling, programming language Java and Netbeans as integrated development environment. The tool allows the creation of installers with a high level of configuration detail. They, guide the installation through detailed wizard where it is not essential that the user has advanced computer skills.

Keywords: GeneSIG, Geographic Information System, installation package.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El desarrollo vertiginoso y constante de la ciencia y la tecnología constituye uno de los pilares fundamentales en los que se sustenta el progreso de la sociedad contemporánea. Uno de los efectos asociados a este proceso es el incremento gradual del volumen de información que se maneja en todos los ámbitos de la vida social actual. El empleo de las Tecnologías de la Información y las Comunicaciones (TIC) favorece el avance exitoso de las actividades realizadas por el hombre en su afán de representar y procesar dicha información. Estas constituyen el catalizador de la sociedad que conocemos y el principal motor que nos acerca a la sociedad de la información y del conocimiento (Roca Chillida, 2010). Uno de los aportes fundamentales de las TIC es lograr la automatización de los procesos que están presentes en disímiles esferas de la sociedad, desde las tareas más sencillas hasta las gestiones más engorrosas. La construcción de sistemas que respondan a estos intereses tributa al ahorro de tiempo y recursos.

Cuba no está exenta de la creación y el uso de estos sistemas y cuenta con varias instituciones dedicadas al desarrollo y comercialización de software. Uno de los centros más comprometidos con esta tarea es la Universidad de las Ciencias Informáticas (UCI), cuya misión es producir aplicaciones y servicios informáticos a partir de la vinculación estudio-trabajo como modelo de formación, además de servir de soporte a la industria cubana de la informática (UCI, 2012). En el centro de desarrollo de software Geoinformática y Señales Digitales (GEySED), perteneciente a la UCI se desarrollan disímiles herramientas que son utilizadas en diversas esferas de la sociedad. Entre las herramientas creadas por el centro se destacan los Sistemas de Información Geográfica (SIG), estos permiten capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada con el fin de resolver problemas complejos de planificación y de gestión apoyándose en el uso de la cartografía digital (Langlé Campos, 2010).

Desde el año 2009 se desarrolla en la UCI una plataforma tecnológica sobre software libre para el desarrollo de SIG en la web denominada GeneSIG, encaminada a realizar la representación y análisis geoespacial de información geográfica. La estructura arquitectónica de la plataforma permite personalizar sus funcionalidades y ajustarlas a cualquier negocio que lo requiera a través de la reutilización de sus componentes. Además GeneSIG puede ser considerado como un SIG Web único y extensible (Pantoja Zaldívar, y otros, 2010). Para el correcto funcionamiento de los SIG desarrollados con la plataforma es

indispensable la instalación de varios paquetes o dependencias que requieren de configuraciones específicas, realizadas actualmente de forma manual por el equipo de desarrollo.

Las personalizaciones de GeneSIG cuentan con una gran variedad de ficheros que contienen los parámetros de conexión con la Base de Datos (BD) de la plataforma y de las capas. A la hora de realizar el despliegue es necesario modificar estos parámetros de conexión y adaptarlos al nuevo entorno del cliente, por lo que la cantidad de ficheros a configurar puede llegar a ser significativa, en dependencia de la complejidad del SIG desarrollado. Un simple cambio de dirección IP en el servidor de BD o una modificación en la clave de acceso a la misma provocan la actualización de todas las capas y ficheros de configuración de la plataforma. Al realizarse manualmente estas acciones se corre el riesgo de omisión de capas por configurar, atentando contra el correcto funcionamiento del sistema y la comunicación entre la plataforma y la BD. Una vez cometidos estos errores resulta engorrosa su identificación y solución, lo que conlleva en ocasiones al reinicio del proceso de instalación, provocando la pérdida de tiempo y esfuerzo.

Partiendo de la problemática planteada anteriormente se establece como **problema a resolver** ¿Cómo agilizar el proceso de instalación y configuración de las personalizaciones de la plataforma GeneSIG?

En correspondencia con el problema, se define como **objeto de estudio** el proceso de instalación y configuración de aplicaciones informáticas.

Para la solución del problema se plantea como **objetivo general:** desarrollar una herramienta que permita generar paquetes de instalación para automatizar el proceso de instalación y configuración de las personalizaciones de la plataforma GeneSIG. En consecuencia el **campo de acción** se enmarca en las operaciones para la instalación y configuración de las personalizaciones de la plataforma soberana GeneSIG.

Para guiar la investigación se definen las siguientes **preguntas científicas**:

- ✓ ¿Cuáles son los fundamentos teóricos para la generación de paquetes de instalación?
- ✓ ¿Cuáles son las características que debe cumplir la herramienta para la generación de paquetes de instalación que permita agilizar el proceso de instalación y configuración de las personalizaciones de la plataforma GeneSIG?

- ✓ ¿Cómo estructurar el proceso de desarrollo de la herramienta para la generación de paquetes de instalación?
- ✓ ¿La herramienta desarrollada permite agilizar el proceso de instalación y configuración de las personalizaciones de la plataforma GeneSIG?

Para dar cumplimiento al objetivo se proponen las siguientes **tareas de la investigación:**

- ✓ Análisis de las diferentes herramientas que se emplean en la creación de paquetes de instalación para obtener información de su funcionamiento así como las características y funcionalidades que presentan.
- ✓ Definición de las herramientas y tecnologías a utilizar para la construcción del sistema.
- ✓ Especificación de los requisitos funcionales y no funcionales a través de entrevistas con el cliente para determinar las cualidades y características que debe poseer la herramienta.
- ✓ Diseño de la propuesta de solución para describir la estructura de la implementación.
- ✓ Implementación de las funcionalidades de la herramienta para dar respuesta a los requisitos.
- ✓ Validación del sistema haciendo uso de pruebas de caja negra para verificar el correcto funcionamiento del mismo.

En el desarrollo del trabajo se emplean los siguientes **métodos científicos:**

Dentro de los métodos teóricos se emplea el analítico-sintético para analizar la bibliografía disponible. Se utiliza con el propósito de extraer y caracterizar los elementos fundamentales que componen el proceso de instalación y configuración de personalizaciones de la plataforma GeneSIG, de forma tal que pueda lograrse una correcta elaboración de la herramienta. El método modelación se utiliza para realizar los diagramas presentes en los artefactos generados durante el proceso de desarrollo de software. Estos diagramas facilitan el trabajo del programador ya que brindan información de gran utilidad para el desarrollo del sistema. El método empírico que se emplea es la entrevista no estructurada para obtener información sobre las funcionalidades que debe poseer la herramienta. Además se usa para identificar las características de la plataforma GeneSIG, qué dependencias requiere la misma y cuáles son los ficheros de configuración que se modifican durante el proceso de despliegue.

El presente trabajo está estructurado de la siguiente forma:

Capítulo 1. Fundamentos teóricos para la generación de paquetes de instalación de personalizaciones de la plataforma GeneSIG: se incluyen los principales elementos asociados al dominio del problema. Se analizan diversos sistemas informáticos que pueden servir de referencia o punto de partida de la presente investigación. Se define la metodología, las herramientas y tecnologías a utilizar para el desarrollo de la aplicación.

Capítulo 2. Análisis y diseño de la herramienta para la generación de paquetes de instalación: se modela el dominio, se realiza la especificación de los requisitos de software y se definen los casos de uso. Se describen la arquitectura y los patrones arquitectónicos y de diseño a emplear. Además se lleva a cabo la creación de los artefactos ingenieriles fundamentales relacionados con el diseño.

Capítulo 3. Implementación y validación de la herramienta para la generación de paquetes de instalación: se realiza la implementación y validación del sistema, se describe el diagrama de componentes y los elementos relacionados con los estándares de codificación y el tratamiento de errores. Además se llevan a cabo diferentes pruebas para verificar el correcto funcionamiento del sistema.

CAPÍTULO 1. FUNDAMENTOS TEÓRICOS PARA LA GENERACIÓN DE PAQUETES DE INSTALACIÓN DE PERSONALIZACIONES DE LA PLATAFORMA GENESIG

El presente capítulo aborda los aspectos teóricos fundamentales para el desarrollo de la solución. Se describe el proceso de instalación de personalizaciones de la plataforma GeneSIG y se analizan diversos sistemas informáticos que pueden servir de referencia o punto de partida de la presente investigación. Se explican los elementos asociados a la investigación y se describen la metodología, las tecnologías y herramientas de software definidas para el proceso de desarrollo.

1.1 Elementos asociados al dominio del problema

Como punto de partida fundamental es necesario el análisis de los elementos asociados al dominio del problema. Estos sientan la base de los fundamentos teóricos y tributan a la comprensión de los elementos principales relacionados con el problema.

1.1.1 GeneSIG

GeneSIG es una plataforma soberana para el desarrollo de SIG sobre la web que presenta una arquitectura basada en componentes, estos componentes pueden ser unidades de modelado, diseño e implementación. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución (Membrides Espinosa, y otros, 2012).

1.1.2 Personalizaciones de GeneSIG

A partir de la plataforma GeneSIG se obtienen las personalizaciones de la misma. Estas, constituyen la reutilización de los componentes de la plataforma ajustándolos a diferentes negocios o actividades desarrolladas por el hombre. Las personalizaciones desarrolladas permiten la manipulación, almacenamiento, captura y análisis de información geográfica apoyando el proceso de toma de decisiones.

1.1.3 Dependencias de software

La plataforma GeneSIG posee un conjunto de dependencias. Una dependencia de software es una aplicación informática o una biblioteca necesaria para el correcto funcionamiento de otro programa. Existen aplicaciones que pueden depender de pocas dependencias o puede llegar a prescindir de grandes y complejos árboles de las mismas.

1.1.4 Servidor de mapas

Dentro de las dependencias requeridas por la plataforma se encuentra el servidor de mapas, este se define como un conjunto de software y hardware utilizado para lograr el acceso y publicación de información geoespacial existente en diferentes formatos (González, y otros, 2009)**.**

Entre las principales ventajas que provee el uso de servidores de mapas se encuentran:

- Acceso a información espacial por parte de un gran número de personas.
- Requerimientos menores de software.
- Facilidad de análisis de datos y atributos espaciales.

1.1.5 Sistema de Gestión de Bases de Datos

Otra de las dependencias de GeneSIG lo constituye el Sistema de Gestión de Bases de Datos (SGBD), el cual es el software que permite la utilización y/o actualización de los datos almacenados en una (o varias) base(s) de datos por uno o varios usuarios. Estos sistemas tienen como objetivos la independencia entre los datos y los programas de aplicación, la integridad, seguridad y recuperación de la información así como la facilidad de manipulación y control centralizado de la misma (Matos García, 1999).

1.1.6 Servidor de aplicaciones web

Al igual que el resto de las dependencias el servidor de aplicaciones web es imprescindible para el funcionamiento de la plataforma. De manera general, es un programa que gestiona cualquier aplicación en el lado del servidor realizando conexiones bidireccionales y/o unidireccionales, así como conexiones síncronas o asíncronas con el cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos se utiliza algún protocolo, generalmente HTTP[1],

[1] Hyper Text Transfer Protocol o Protocolo de Transferencia de Hipertexto.

perteneciente a la capa de aplicación del modelo OSI[2]. El término también se emplea comúnmente para referirse al ordenador que ejecuta el programa (Vega Silva, 2011).

1.1.7 Paquetes de instalación

Los paquetes de instalación son la agrupación instalable de los diferentes archivos necesarios para que un programa funcione. Contienen una distribución de la aplicación y los archivos de configuración, tanto del proceso de instalación, como de la configuración inicial de la aplicación.

Muchas veces esos archivos tienen que estar colocados en diferentes partes del sistema y en ocasiones relacionados con otros archivos. Todas estas tareas se pueden hacer manualmente, pudiendo resultar complejas y engorrosas. Por lo general en la mayoría de los casos se utiliza un instalador, que no es más que un programa especial que realiza todas esas tareas de manera automática. Algunos presentan la opción de instalar solamente algunos elementos mientras otros, permiten personalizar la instalación y realizar configuraciones más específicas de la aplicación (Lasso, 2008).

1.2 Actividades que se desarrollan en el despliegue de personalizaciones de la plataforma soberana GeneSIG

Para desplegar personalizaciones de la plataforma GeneSIG es necesario desarrollar un conjunto de operaciones con el objetivo de preparar el entorno donde se va a instalar. Estas operaciones se definen a continuación.

1. **Instalación y configuración del SGBD:** se comprueba la existencia del gestor PostgreSQL y su configuración, en caso de no estar instalado se procede a la instalación y configuración del mismo. Se verifica además la existencia de la extensión PostGIS y se procede a crear las bases de datos y restaurar los backups asociados a las mismas.
2. **Instalación del servidor de mapas:** se comprueba la existencia del servidor de mapas Mapserver, en caso de no existir se procede a su instalación y configuración.

[2] Open System Interconnection o Sistema de Interconexión Abierto.

3. **Instalación del servidor de aplicaciones web:** al igual que el resto de las dependencias primero debe verificarse si se encuentra presente en el ordenador. En caso de no estar instalado Apache, se realiza la instalación y configuración del mismo y de todas las bibliotecas de PHP[3]5 necesarias.
4. **Configuración de la personalización desarrollada:** una vez instaladas las dependencias se está en condiciones de configurar la personalización, para ello lo primero es extraer dicha personalización en el directorio raíz del servidor de aplicaciones web. Posteriormente se editan los ficheros donde están los parámetros de conexión con la base de datos de la plataforma y se establecen los nuevos valores que serán empleados en la comunicación con el gestor de base de datos. Esta acción también se realiza para cada una de las capas vectoriales lo que puede convertirse en un proceso engorroso debido a la complejidad del sistema desarrollado. Todas estas modificaciones son realizadas de forma manual por lo que la ocurrencia de configuraciones erróneas se incrementa significativamente.

1.2.1 Estructura física de la plataforma

Para lograr una mejor comprensión de la plataforma soberana GeneSIG se realiza un análisis de la estructura física que esta posee.

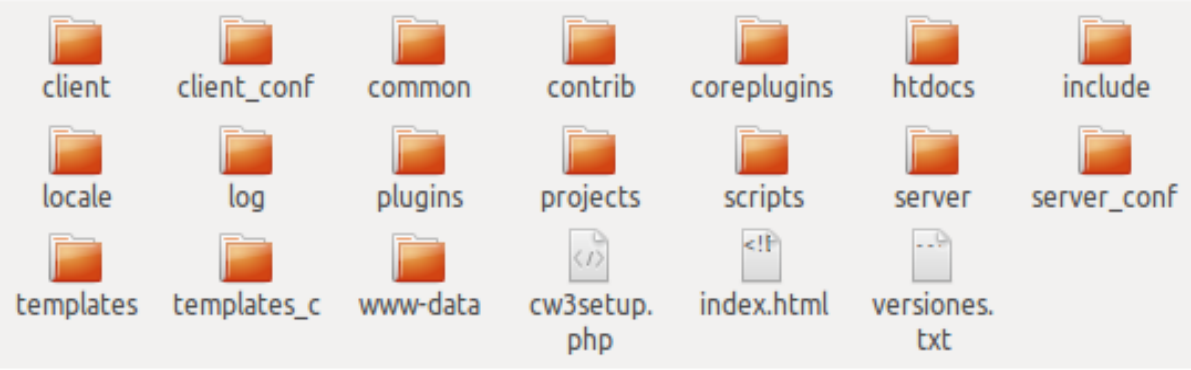


Fig. 1: Estructura física que posee la plataforma GeneSIG.

[3]Hypertext Pre-processor.

Tabla 1: Descripción de los directorios que componen GeneSIG.

Directorio	**Descripción**
client	Directorio para almacenar las clases bases encargadas de la comunicación cliente-servidor.
client_conf	Directorio en que se alojan los ficheros de configuración de la parte cliente.
common	Directorio en que se almacenan las clases bases que constituyen el soporte arquitectónico tanto para la parte cliente como servidor.
contrib	Directorio donde se almacenan algunas librerías necesarias para la plataforma.
coreplugins	Directorio donde se almacenan los coreplugins o componentes que tienen influencia directa y de carácter horizontal para el resto de la plataforma.
htdocs	Directorio de publicación.
include	Directorio en que son alojadas las librerías externas a la plataforma.
locate	Directorio que contiene ficheros locales productos de la instalación del sistema.
log	Directorio en que son alojados los diferentes mensajes que permiten tener un control y trazabilidad de errores.
plugins	Directorio en que se encuentran los plugins de carácter general y que pueden ser reutilizados en varios proyectos.
projects	Directorio en que se almacenan los distintos proyectos o aplicativos dirigidos a negocios específicos de un entorno determinado.
scripts	Directorio que contiene disímiles ficheros script relacionados con la instalación del marco de trabajo.
server	Directorio en que se almacenan las clases bases, responsables de la interacción con el servidor de mapas y publicación de servicios.
server_conf	Directorio en que se alojan los ficheros de configuración de la parte servidora.

templates	Directorio en que se almacenan las plantillas de propósito general, las cuales pueden ser redefinidas por los distintos proyectos.
cw3setup.php	Fichero script que contiene el listado de instrucciones orientadas a la instalación y administración de los recursos de la plataforma.

1.3 Análisis de las soluciones existentes para la generación de instaladores

Para lograr una mejor comprensión de las herramientas más usadas para la generación de paquetes de instalación se realiza un análisis detallado de las características, propiedades y funcionalidades que estas presentan.

1.3.1 Ubucompilator

Ubucompilator es un sencillo programa realizado con Gambas, un lenguaje de programación derivado de BASIC que permite generar paquetes .deb a partir del código fuente de los programas. Este proceso se reduce a unos cuantos pasos a través de una interfaz gráfica, al final del mismo se dispone de un paquete instalable compatible con Debian y sus distribuciones derivadas como son: Ubuntu, Linux Mintt, Super OS. Se distribuye bajo licencia GPL[4] v3 (Saiz, 2010).

1.3.2 Installer Vise

Funciona como una especie de paquete para aplicaciones, permite empaquetar múltiples ficheros en un solo instalador y así, distribuir fácilmente paquetes de programas a través de Internet. Los instaladores pueden personalizarse al antojo del cliente, indicando al programa cuales son las áreas específicas del sistema donde se han de instalar los distintos ficheros. Soporta los sistemas Mac OS 8 y Mac OS 9, así como Mac OS X (Digital River Inc, 2014).

1.3.3 Mep Installer

[4] General Public License o Licencia Pública General.

Es una herramienta que permite crear paquetes de instalación. Dentro de las principales características de los instaladores que genera se encuentra que chequea y detecta si hay instalaciones previas del programa, además cuenta con un asistente personalizado para que el usuario pueda ir realizando paso a paso la instalación. La herramienta posee licencia freeware y tiene la capacidad de comprimir todos los ficheros del programa en un único archivo ejecutable .exe. Brinda la opción de desinstalación por si el usuario desea eliminar el programa del ordenador. Los paquetes de instalación pueden ser creados con soporte para distintos lenguajes y permiten la creación de accesos directos en cualquier lugar, incluyendo en el menú Inicio y en el escritorio (Mep Producciones, 2007).

1.3.4 IzPack

IzPack es un generador de instaladores basado en Java y su distribución se realiza bajo la licencia Apache Software License 2.0 (García Pérez, 2006)**.** Genera instaladores multiplataforma en un único archivo .jar dependiendo de que el sistema operativo tenga instalado el JRE[5], el cual es el requerimiento mínimo para poder ejecutar una aplicación Java. IzPack en sí es una utilidad de línea de comandos, pero existe una aplicación que hace de front-end basada en Java Swing (Mesa Rodríguez, 2011)**.**

Tabla 2: Comparación entre las herramientas para la creación de instaladores.

Herramienta	**Licencia**	**Plataforma de desarrollo**	**Plataforma de producción**	**Tipo de archivos**
Installer Vise	Comercial	Mac	Mac	Script
Mep Installer	Freeware	Windows	Windows	Script
Ubucompilator	GPL	Linux	Linux	Script
IzPack	Apache Software License 2.0	Windows y Linux	Windows y Linux	XML[6]

[5] Java Runtime Enviroment o Ambiente de Ejecución de Java.

[6] eXtensible Markup Language o Lenguaje de Etiquetado Extensible.

Teniendo en cuenta que la plataforma GeneSIG es desarrollada con herramientas Open Source y que las personalizaciones de la misma serán desplegadas sobre plataformas GNU/Linux el análisis se reduce solo a las herramientas IzPack y Ubucompilator.

En el caso de IzPack, una de sus principales limitantes es que los paquetes de instalación generados se empaquetan en un archivo .jar necesitando del JRE para su funcionamiento lo que constituiría otra dependencia más para la personalización. Por su parte, la herramienta Ubucompilator solamente toma el conjunto de dependencias y genera un paquete con las mismas, de esta forma no permite la interacción con el usuario ni la especificación de configuraciones durante la ejecución del instalador generado.

Una vez realizado este análisis se decide crear una herramienta que se ajuste a las exigencias de las personalizaciones de GeneSIG. Esta debe ser totalmente configurable y los paquetes de instalación generados con dicha herramienta deben permitir además de la instalación de la personalización, la edición futura de las configuraciones de la misma.

Aunque no fue seleccionada ninguna de las herramientas analizadas, se tuvieron en cuenta algunas de sus características para la creación del nuevo generador de paquetes de instalación, como son: el uso de ficheros XML para las configuraciones y la compresión de ficheros y dependencias. Otra de las características a tener en cuenta es que el instalador obtenido debe chequear y detectar si existen instalaciones previas de las dependencias, además de contar con un asistente personalizado para guiar el proceso de instalación.

1.4 Metodología de desarrollo de software

Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo. Existen dos clasificaciones de metodologías de desarrollo de software, las tradicionales y las ágiles.

Las metodologías tradicionales o pesadas llevan una documentación bien detallada del proyecto y se enfocan en el cumplimiento de un plan que se define en la fase inicial del mismo. Se caracterizan por ser muy costosas ante la presencia de cambios y no permiten cierta flexibilidad en proyectos donde el entorno es cambiante.

A su vez las metodologías ágiles surgen como respuesta a los problemas que puedan ocasionar las metodologías tradicionales y basan su fundamento en la adaptabilidad de los procesos de desarrollo. Estas metodologías ponen de relevancia que la capacidad de respuesta es más importante que el seguimiento estricto de un plan (Inteco, 2009).

En (Inteco, 2009) se estable una comparación entre estos grupos de metodologías:

Tabla 3: Comparación entre las metodologías ágiles y tradicionales.

Metodologías ágiles	**Metodologías tradicionales**
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas y normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (generalmente menos de diez integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Después de analizar las principales características de cada grupo de metodologías y teniendo en cuenta que el proyecto a desarrollar puede cambiar a medida que va evolucionando, el contrato es bastante flexible y el equipo de trabajo es reducido y con poca experiencia se decide emplear una metodología ágil.

Dentro de las metodologías de desarrollo de software ágiles, se emplea Open UP ya que cuenta con procesos de baja formalidad y estos pueden ampliarse para hacer frente a una gran variedad de proyectos, se centra en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo. Además Open UP es una metodología mínimamente suficiente, esto quiere decir que incluye solo el contenido fundamental. No provee orientación sobre temas en los que el proyecto tiene que lidiar, como son: el tamaño del equipo, el cumplimiento, seguridad, orientación tecnológica entre otras, sin embargo es completa en el sentido de que manifiesta a plenitud el proceso de construir un sistema (Olguin Juarez, 2013).

Otros de los beneficios de la metodología es que permite detectar errores tempranos a través de un ciclo iterativo y es apropiada para proyectos pequeños donde es escasa la disponibilidad de recursos. Otro de los aspectos fundamentales de Open UP es que disminuye la probabilidad de fracaso e incrementa las probabilidades de éxito de los proyectos pequeños; esta posee un enfoque centrado al cliente y evita la elaboración de documentación y diagramas requeridos en la metodología RUP[7].

Otras características de Open UP que revisten particular importancia son:

- Desarrollo incremental.
- Uso de casos de uso y escenarios.
- Manejo de riesgos.
- Diseño basado en la arquitectura.

Principios de Open UP

- **Colaborar para sincronizar intereses y compartir conocimiento**: promueve prácticas que impulsan un ambiente de equipo saludable, facilitan la colaboración y desarrollan un conocimiento compartido del proyecto.

[7] Rational Unified Process.

- **Equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto:** promueve prácticas que permiten a los participantes de los proyectos desarrollar una solución que maximice los beneficios obtenidos por los participantes y que cumple con los requisitos y restricciones del proyecto.
- **Desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo:** promueve prácticas que permiten a los equipos de desarrollo obtener retroalimentación temprana y continua de los participantes del proyecto.

Fases de Open UP

- **Concepción**: en esta fase se tienen en cuenta las necesidades de cada participante del proyecto en los objetivos del ciclo de vida. Se definen para el proyecto los límites, el criterio de aceptación y se analiza a grandes rasgos la planificación.
- **Elaboración**: en esta fase se realiza un análisis del dominio, se define la arquitectura del sistema y se analizan los riesgos significativos. Se confecciona un plan del proyecto donde se definen los requisitos y el proceso de desarrollo.
- **Construcción**: esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la arquitectura definida.
- **Transición**: es la última fase y su propósito es asegurar que el sistema es entregado a los usuarios finales. Se evalúa la funcionalidad del último entregable de la fase de construcción y se desarrolla la capacitación de los usuarios y del personal encargado del mantenimiento del sistema.

1.5 Lenguaje de modelado

Se utiliza UML[8] 2.0 como lenguaje de modelado en la construcción y documentación de los artefactos. No es un lenguaje de programación y no tiene propietario, está basado en el común acuerdo de gran parte de la comunidad informática (Rumbaugh, y otros, 2000). No guía al desarrollador en la forma de realizar el análisis y diseño, ni le indica cuál proceso de desarrollo de software adoptar. Una de las ventajas que posee

[8] Unified Modeling Language o Lenguaje de Modelado Unificado.

es que diseñadores diferentes, modelando sistemas diferentes, pueden entender cada uno los diseños de los otros. Este lenguaje se emplea en la fase de análisis y diseño de la herramienta en la realización de los diferentes diagramas. Su utilización se evidencia a través de la herramienta interactiva de modelado visual Visual Paradigm.

1.6 Lenguaje de programación

El lenguaje de programación que se utiliza es la versión 1.7 de Java, este incorpora muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (ejecución remota, componentes, acceso a bases de datos y otros). Por esta razón muchos expertos opinan que Java es el lenguaje ideal para aprender la informática moderna, ya que incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro. La compañía Sun MicroSystems describe el lenguaje Java como *"simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico"* (Rodríguez, y otros, 2000). Otro de los principales aspectos presentes en la elección del lenguaje es que el equipo de desarrollo tiene conocimiento y dominio del mismo.

Ventajas de Java

- **Multiplataforma:** se ejecuta en la mayoría de los sistemas operativos, incluyendo sistemas operativos móviles.
- **Software de distribución libre:** no es necesario pagar una licencia para poder comenzar a desarrollar en este lenguaje.
- **Orientado a objetos:** el paradigma de programación orientada a objetos es capaz de acercar la forma de programar a la forma de pensar del ser humano.
- **Gestión de memoria:** la liberación de memoria se hace de manera automática lo cual es un proceso trasparente al programador.
- **Programación multi-hilos:** brinda un enfoque elegante y fácil de emplear basado en bloques sincronizados.

1.7 Tecnologías y herramientas empleadas

Se identificaron un conjunto de tecnologías y herramientas que determinan la construcción del sistema así como las características y propiedades que este debe poseer. A continuación se explican cuáles son estas tecnologías y herramientas medulares en el desarrollo de la solución.

1.7.1 JDK 1.7

El software Java Development Kit (JDK) provee un conjunto de herramientas, utilidades, documentación y ejemplos para desarrollar aplicaciones Java. Puede ser instalado en una computadora local o en una unidad de red. Incluye aparte del JRE un compilador y el empaquetador de archivos .jar (Latorre, 2010). El JRE es el requerimiento mínimo de software que debe presentar un equipo para poder ejecutar aplicaciones desarrolladas en Java. Está compuesto por las clases que conforman la API[9] y la Máquina Virtual de Java (JVM, por sus siglas en inglés). Dentro de las tareas principales realizadas por la JVM se encuentran: la reservación de memoria para los objetos creados, la liberación de memoria no usada y las llamadas al sistema huésped como es el caso del acceso a los diferentes dispositivos.

1.7.2 XML 1.0

En los ficheros de configuración se emplea XML que se caracteriza por ser muy simple y a la vez estricto, juega un papel fundamental en el intercambio de datos entre las aplicaciones. Es un lenguaje muy similar a HTML[10] pero su función principal es describir datos y no mostrarlos. XML es un formato que al ser utilizado permite la comunicación entre diferentes aplicaciones (W3C, 2010). Es empleado para estructurar, almacenar e intercambiar información. Se emplea además ya que muchos de los ficheros de configuración de la plataforma GeneSIG presentan este formato, tal es el caso del fichero modulesconfig.xml el cual contiene los parámetros de conexión a la BD de la plataforma.

1.7.3 Visual Paradigm for UML 8.0

Visual Paradigm for UML 8.0 es una herramienta CASE[11] multiplataforma que emplea UML como lenguaje de modelado y ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite construir diagramas de diversos tipos, código inverso, generar código desde diagramas y generar

[9] Applications Programming Interface o Interfaz de Programación de Aplicaciones.

[10] Hyper Text Markup Language.

[11] Computer Aided Software Engineering o Ingeniería de Software Asistida por Computadora.

documentación. La herramienta también proporciona abundantes tutoriales, demostraciones interactivas y proyectos UML (Cabrera, y otros, 2010).

1.7.4 Netbeans IDE 7.4

NetBeans es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés), modular y de código abierto. El proyecto NetBeans fue fundado por Sun MicroSystems en junio de 2000 y es aún el patrocinador principal de los proyectos. Una de las características del IDE es que asiste al programador de forma parcial en la escritura de código, aunque no lo libera de aprender el lenguaje de programación. Permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Debido a que estos módulos pueden ser desarrollados independientemente, las aplicaciones basadas en NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software (Netbeans Team, 2011). Es fácil de instalar y de uso instantáneo, además se ejecuta en varias plataformas, incluyendo Windows, Linux y Mac OS X y Solaris.

1.7.5 Geany 1.22

Geany es un editor de texto con algunas funciones básicas integradas, que facilitan la labor de programación al usuario. Las funciones que más se destacan son la corrección de sintaxis, el plegado de código, autocerrado de tags y la edición simultánea de varios archivos en distintas pestañas. Brinda soporte para C, Java, PHP, HTML, Python, Perl y Pascal (UptoDown, 2013). Esta herramienta se utiliza para la creación y edición de los diferentes scripts que controlan el proceso de instalación y configuración.

1.7.6 Gimp 2.6

Para el diseño de las interfaces y la iconografía del sistema se utiliza GIMP (GNU Image Manipulation Program), que es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito distribuido bajo la licencia GPL. Existen versiones totalmente funcionales para Windows y Mac OS X, además de estar disponible en varios idiomas. Surgió como una herramienta libre para el tratamiento de imágenes y se ha convertido en una alternativa libre y eficaz al Photoshop (Gimp, 2013).

Conclusiones del capítulo

El análisis de las características y funcionalidades de diversas herramientas para la creación de instaladores arrojó que las mismas presentan limitaciones que impiden que sean utilizadas para dar solución al problema, de esta forma se hace necesario desarrollar una herramienta para la generación de paquetes de instalación que cumpla con las exigencias del despliegue de las personalizaciones de GeneSIG. Se adopta Java 1.7 como lenguaje de programación, NetBeans como IDE y Open UP como metodología de desarrollo.

CAPÍTULO 2. ANÁLISIS Y DISEÑO DE LA HERRAMIENTA PARA LA GENERACIÓN DE PAQUETES DE INSTALACIÓN

En el presente capítulo se elabora el modelo de dominio y se exponen los conceptos y relaciones que intervienen en el mismo. Se realiza la especificación de los requisitos de software y se definen los casos de uso. Se describen los elementos de la arquitectura y los patrones arquitectónicos y de diseño a emplear. Se crean los diagramas correspondientes al modelo de diseño que son de vital importancia para la implementación del sistema.

2.1 Modelo de dominio

EL modelo de dominio captura los tipos más importantes de objetos en el contexto del sistema, estos objetos representan las cosas que existen o los eventos que suceden en el entorno del sistema (Rumbaugh, y otros, 2000). Teniendo en cuenta que los procesos de negocio no se pueden determinar con facilidad y las fronteras no se encuentran bien definidas, además es difícil establecer reglas de funcionamiento se decide emplear el modelo de dominio. Este puede ser considerado como el punto de partida para el diseño de la solución.

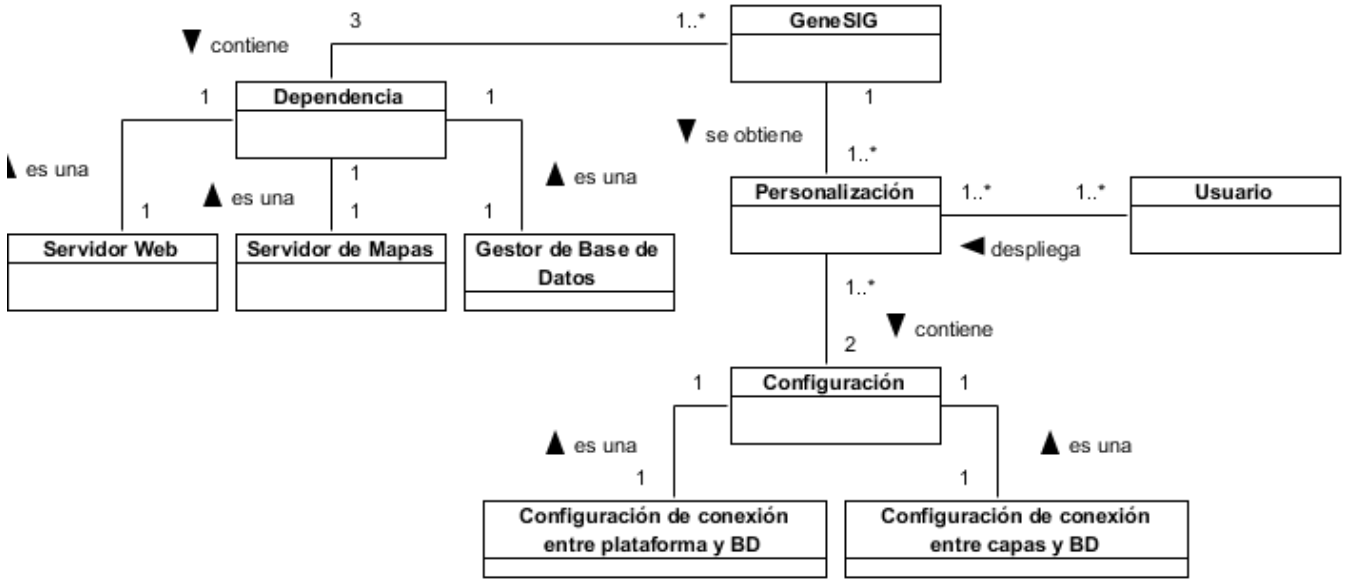


Fig. 2: Diagrama de clases del dominio.

2.1.1 Glosario de términos del dominio

Para lograr una mejor comprensión del modelo de dominio se explican brevemente las clases fundamentales presentes en el mismo.

GeneSIG: plataforma para el desarrollo de SIG, esta posee una arquitectura basada en componentes.

Personalización de GeneSIG: reutilización de los componentes de la plataforma ajustándolos a diferentes negocios o actividades.

Dependencia de software: aplicación o biblioteca necesaria para el correcto funcionamiento de otro programa.

Servidor de mapa: proveedor de cartografía a través de la red tanto en modo raster como vectorial.

Sistema de Gestión de Bases de Datos: software que permite la utilización y/o la actualización de los datos almacenados en una o varias base(s) de datos.

Servidor de aplicaciones web: programa que gestiona cualquier aplicación en el lado del servidor generando respuestas a las solicitudes de los clientes.

Configuración: diferentes parámetros que se establecen para el correcto funcionamiento de la personalización.

Configuración de conexión entre plataforma y BD: conjunto de parámetros que se establecen para la conexión entre la plataforma y la BD.

Configuración de conexión entre capas y BD: conjunto de parámetros que se establecen para la conexión entre las capas y la BD.

Usuario: persona que interactúa con el sistema durante el proceso de despliegue.

2.1.2 Descripción del diagrama de clases del dominio

La plataforma soberana GeneSIG presenta un conjunto de dependencias que son imprescindibles para su correcto funcionamiento. Entre las dependencias se encuentran el servidor de aplicaciones web, el servidor

de mapas y el gestor de base de datos. A través de la reutilización de los componentes de la plataforma se pueden obtener una o varias personalizaciones de la misma, estas presentan diferentes configuraciones como son la configuración de conexión entre la plataforma y la BD y la configuración de conexión entre la las capas y la BD. Una vez realizada la configuración de la personalización y disponiendo de las dependencias necesarias es posible realizar el despliegue con el cliente, proceso que es realizado de forma manual por el usuario.

2.2 Requisitos del sistema

Un requisito de software es una condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente (Rumbaugh, y otros, 2000). Los requisitos se clasifican en funcionales y no funcionales.

2.2.1 Requisitos funcionales

Los Requerimientos Funcionales (RF) son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. Los RF describen lo que el sistema debe hacer (Sommerville, 2005). A continuación se describen los requisitos funcionales de la solución.

RF 1: Cargar personalización

Esta funcionalidad permite cargar una personalización de la plataforma GeneSIG a la cual se le va a generar un paquete de instalación. El usuario debe introducir la ruta correspondiente a la ubicación de la personalización y la ruta de salida del paquete de instalación, por lo que se necesitan los siguientes criterios de entrada:

- Ruta de la personalización a cargar (Formato: Alfanumérico, Obligatorio: Sí).
- Ruta de salida del paquete de instalación a generar (Formato: Alfanumérico, Obligatorio: Sí).
- Nombre del paquete de instalación a generar (Formato: Alfanumérico, Obligatorio: Sí).

Los criterios de salida son los siguientes:

- Visualización de las configuraciones de la personalización cargada.

- Información general de la personalización.
- Dependencias de software disponibles para la generación del paquete de instalación.

RF 2: Modificar parámetros de conexión de la plataforma

Esta funcionalidad permite modificar los parámetros de conexión de la base de datos que utiliza la plataforma. Se necesitan los siguientes criterios de entrada:

- Formato (Formato: Alfanumérico, Obligatorio: Sí).
- Gestor de Base de Datos (Formato: Alfanumérico, Obligatorio: Sí).
- Host (Formato: Alfanumérico, Obligatorio: Sí).
- Usuario (Formato: Alfanumérico, Obligatorio: Sí).
- Clave (Formato: Alfanumérico, Obligatorio: Sí).
- Base de Datos (Formato: Alfanumérico, Obligatorio: Sí).
- Puerto (Formato: Numérico, Obligatorio: Sí).

El criterio de salida es el siguiente:

- Modificación de los parámetros de conexión de la base de datos de la plataforma.

RF 3: Modificar parámetros de conexión de las capas vectoriales

Esta funcionalidad permite modificar los parámetros de conexión de la base de datos de las capas vectoriales. Se necesitan los siguientes criterios de entrada:

- Formato (Formato: Alfanumérico, Obligatorio: Sí).
- Gestor de Base de Datos (Formato: Alfanumérico, Obligatorio: Sí).
- Host (Formato: Alfanumérico, Obligatorio: Sí).
- Usuario (Formato: Alfanumérico, Obligatorio: Sí).
- Clave (Formato: Alfanumérico, Obligatorio: Sí).
- Base de Datos (Formato: Alfanumérico, Obligatorio: Sí).
- Puerto (Formato: Numérico, Obligatorio: Sí).

El criterio de salida es el siguiente:

- Modificación de los parámetros de conexión de la base de datos de las capas vectoriales.

RF 4: Modificar selección de las dependencias

Esta funcionalidad permite modificar la selección de las dependencias de software disponibles. Se necesitan los siguientes criterios de entrada:

- Distribución (Formato: Campo de selección, Obligatorio: Sí).
- Plataforma (Formato: Campo de selección, Obligatorio: Sí).
- Servidor de mapas (Formato: Campo de selección, Obligatorio: Sí).
- Servidor de BD (Formato: Campo de selección, Obligatorio: Sí).
- Servidor de aplicaciones Web (Formato: Campo de selección, Obligatorio: Sí).

El criterio de salida es el siguiente:

- Actualización de la selección de las dependencias del instalador.

RF 5: Configurar host virtual

Esta funcionalidad permite configurar el host virtual de la personalización de la plataforma GeneSIG a la cual se le va a generar un paquete de instalación. Se necesita el siguiente criterio de entrada:

- Dominio (Formato: Alfanumérico, Obligatorio: Sí).

El criterio de salida es el siguiente:

- Se actualizan los parámetros para la configuración del host virtual.

RF 6: Crear fichero de configuración

Esta funcionalidad permite crear ficheros de configuración de la personalización de la plataforma GeneSIG. Se necesitan los siguientes criterios de entrada:

- Ruta de creación (Formato: Alfanumérico, Obligatorio: Sí).
- Nombre del fichero (Formato: Alfanumérico, Obligatorio: Sí).
- Tipo (Formato: Alfanumérico, Obligatorio: Sí).

El criterio de salida es el siguiente:

- Se crea el fichero de configuración en la ruta especificada.

RF 7: Eliminar fichero de configuración

Esta funcionalidad permite eliminar ficheros de configuración de la personalización de la plataforma GeneSIG. Se necesita el siguiente criterio de entrada:

- Ruta del fichero (Formato: Alfanumérico, Obligatorio: Sí).

El criterio de salida es el siguiente:

- Se elimina el fichero de configuración especificado.

RF 8: Editar fichero de configuración

Esta funcionalidad permite la edición de ficheros de configuración de la personalización de la plataforma GeneSIG. Se necesitan el siguiente criterio de entrada:

- Ruta del fichero (Formato: Alfanumérico, Obligatorio: Sí).

El criterio de salida es el siguiente:

- Se muestra el contenido del fichero de configuración posibilitando su edición.

RF 9: Probar conexión a BD

Esta funcionalidad permite conocer el estado de conexión con las BD. Se necesitan los siguientes criterios de entrada:

- Host (Formato: Alfanumérico, Obligatorio: Sí).
- Usuario (Formato: Alfanumérico, Obligatorio: Sí).
- Clave (Formato: Alfanumérico, Obligatorio: Sí).
- Base de Datos (Formato: Alfanumérico, Obligatorio: Sí).
- Puerto (Formato: Numérico, Obligatorio: Sí).

El criterio de salida es el siguiente:

- Se muestra el estado de conexión con las BD.

RF 10: Generar paquete de instalación

Esta funcionalidad permite generar un paquete de instalación a partir de una personalización, dicho paquete contiene las dependencias necesarias y la configuración inicial de la personalización. Se necesita el siguiente criterio de entrada:

- Personalización de GeneSIG (Formato: Alfanumérico, Obligatorio: Sí).

El criterio de salida es el siguiente:

- El paquete de instalación asociado a la personalización y todas sus dependencias.

2.2.2 Requisitos no funcionales

Los Requisitos No Funcionales (RNF) son restricciones de los servicios o funciones ofrecidas por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los RNF especifican o restringen las propiedades emergentes del sistema (Sommerville, 2005).

Requisitos de usabilidad

- El sistema debe presentar una interfaz sencilla y amigable, contando con una arquitectura de la información intuitiva y de fácil acceso a las funcionalidades.
- Debe caracterizarse por el empleo de iconos sugerentes al usuario así como contener una sección de ayuda donde se expliquen los aspectos de mayor interés y complejidad del sistema.
- Debe admitir flujos alternos, como cancelar la operación y salir de la vista actual.
- En caso de producirse errores se indicará de manera constructiva la posible solución.

Portabilidad

- La solución podrá emplearse en varios sistemas operativos como son Windows y Linux en sus versiones (XP, Server 2008, Vista, 7 y 8) y (Ubuntu 12.04 y superior, Debian 12.04 y superior) respectivamente.

- Los paquetes de instalación generados con la solución deben funcionar correctamente en las distribuciones de Linux (Debian 12.04 y superior) y (Ubuntu 12.04 y superior).

Software

- Deberá estar instalado el JRE en la estación de trabajo donde se vaya a emplear la herramienta para la generación de paquetes de instalación.

Hardware

- El sistema podrá utilizarse en estaciones de trabajo que posean como mínimo 512 MB de memoria RAM[12], microprocesador Intel® Dual-Core o Intel® Core-2 Duo a 1.8 GHz y una capacidad de 40 GB de disco duro.

2.3 Descripción del sistema

A continuación se describe el diagrama de casos de uso del sistema y se identifican los casos de uso arquitectónicamente significativos.

2.3.1 Diagrama de casos de uso del sistema

Un diagrama de casos de uso constituye una representación del contexto del sistema. Muestra los límites del mismo, lo que permanece fuera de él, y cómo se utiliza. Sirve como herramienta de comunicación que resume el comportamiento de un sistema y sus actores (Larman, 2002).

[12] Random Access Memory o Memoria de Acceso Aleatorio.

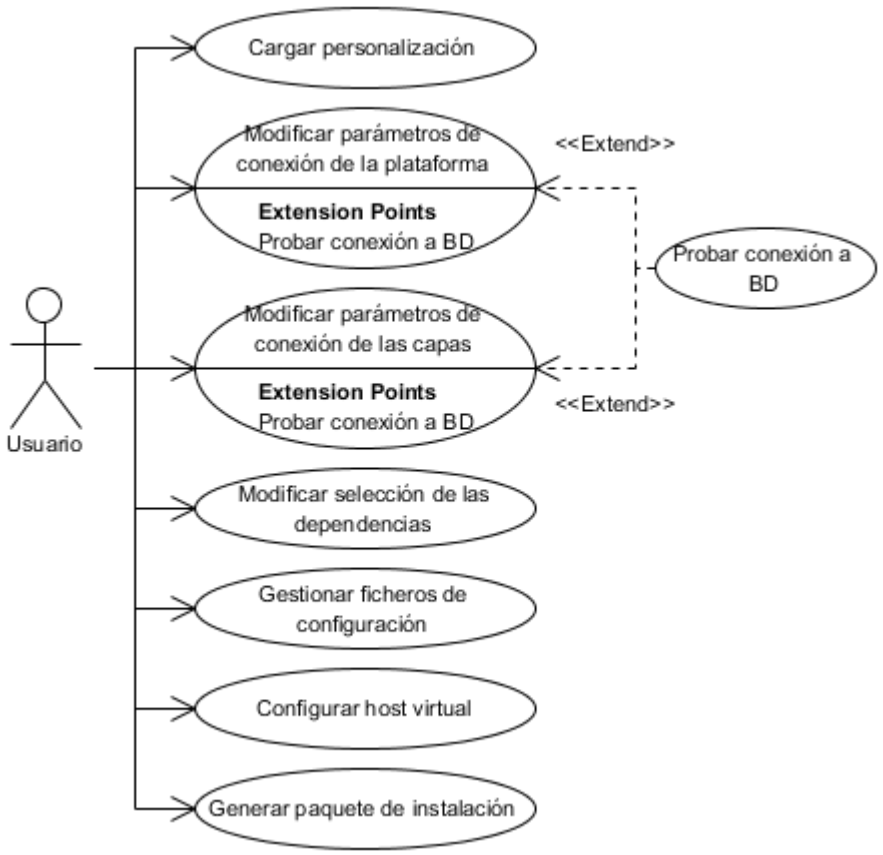


Fig. 3: Diagrama de casos de uso del sistema.

2.3.2 Casos de uso arquitectónicamente significativos

Los casos de usos arquitectónicamente significativos son aquellos que contribuyen a mitigar los riesgos más importantes, además se caracterizan por ser imprescindibles para los usuarios del sistema y ayudan a cubrir las funcionalidades significativas. Representan las partes más críticas de la arquitectura del sistema.
Los casos de uso arquitectónicamente significativos presentes en la propuesta de solución son:

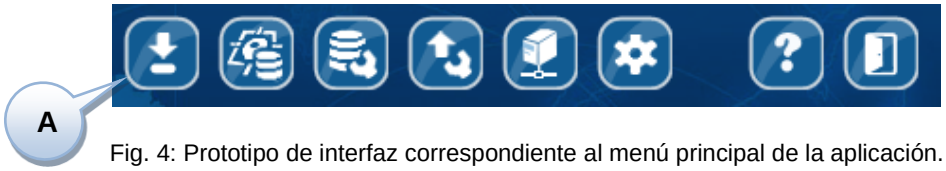
- Caso de uso “Cargar personalización”
- Caso de uso “Generar paquete de instalación”

2.3.3 Descripción de los casos de uso del sistema

A continuación se describen los casos de uso del sistema “Cargar personalización” y “Generar paquete de instalación”, el resto de las descripciones se encuentran en el documento “SOLUSHION Modelo del Sistema v2.0.doc” perteneciente a los artefactos ingenieriles de la investigación.

Tabla 4: Descripción del caso de uso “Cargar personalización”.

Caso de Uso:	Cargar personalización.
Actores:	Usuario
Propósito:	Este caso de uso se realiza con el objetivo de que el usuario pueda cargar una personalización de GeneSIG.
Resumen:	El caso de uso se inicia cuando el actor selecciona la opción “Cargar Personalización”. El caso de uso termina una vez finalizada alguna de las opciones que permite realizar.
Precondiciones:	El sistema debe estar ejecutándose correctamente.
Referencias:	RF 1.
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor Usuario selecciona la opción “Cargar Personalización” (A), como se muestra en la Interfaz 1.	**2.** El sistema muestra una ventana solicitando los datos necesarios para cargar una personalización. Ver interfaz 2.
3. El actor Usuario introduce la ruta de la personalización, para lograr esto selecciona la opción “Abrir” (C) como se muestra en la Interfaz 2.	**4.** El sistema muestra una ventana de selección de archivos. Ver interfaz 3.

5. El actor Usuario especifica el directorio de la personalización y selecciona la opción “Seleccionar” (J).	**6.** El sistema guarda la ruta, oculta la ventana de selección de archivos y actualiza el campo (B) en la Interfaz 2.
7. El actor Usuario introduce la ruta de destino del instalador para lograr esto selecciona la opción “Abrir” (E) como se muestra en la Interfaz 2.	**8.** El sistema muestra una ventana de selección de archivos. Ver interfaz 3.
9. El actor Usuario especifica el directorio de salida del instalador y selecciona la opción “Seleccionar” (J).	**10.** El sistema guarda la ruta, oculta la ventana de selección de archivos y actualiza el campo (D) en la Interfaz 2.
11. El actor Usuario introduce el nombre del directorio del instalador (F) y selecciona la opción “Cargar” (G).	**12.** El sistema comprueba que no existan campos vacíos y que los datos introducidos sean correctos. Carga la personalización y visualiza los datos cargados, terminando así el caso de uso.
Prototipos de Interfaz	
Interfaz 1	
 Fig. 4: Prototipo de interfaz correspondiente al menú principal de la aplicación.	
Interfaz 2	

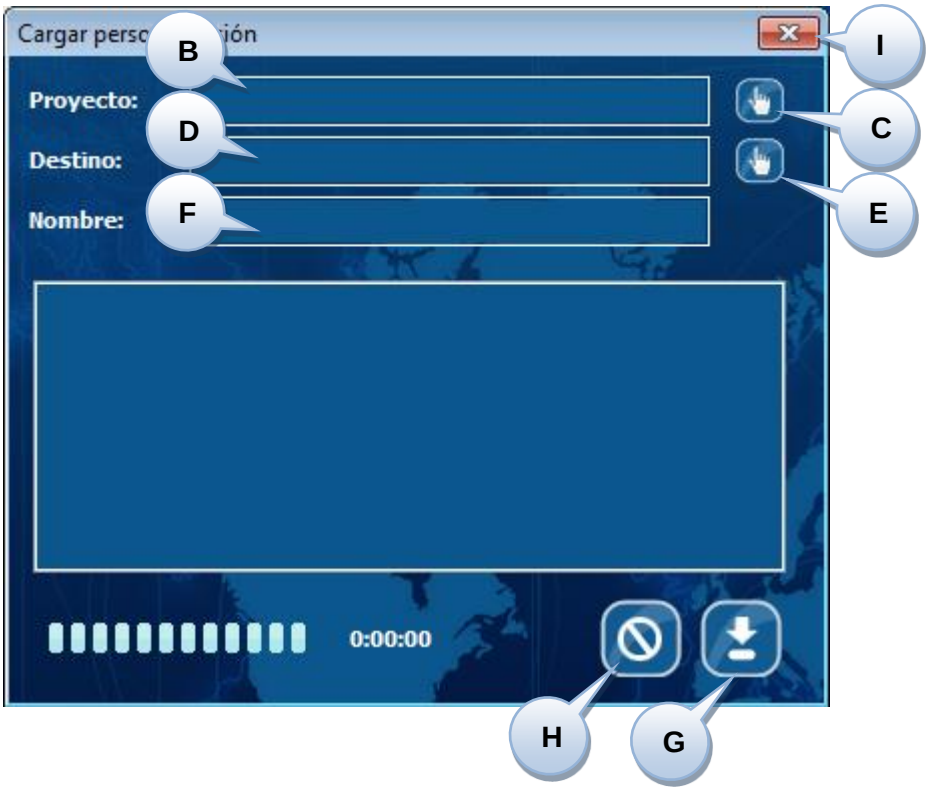


Fig. 5: Prototipo de interfaz “Cargar personalización”

Interfaz 3

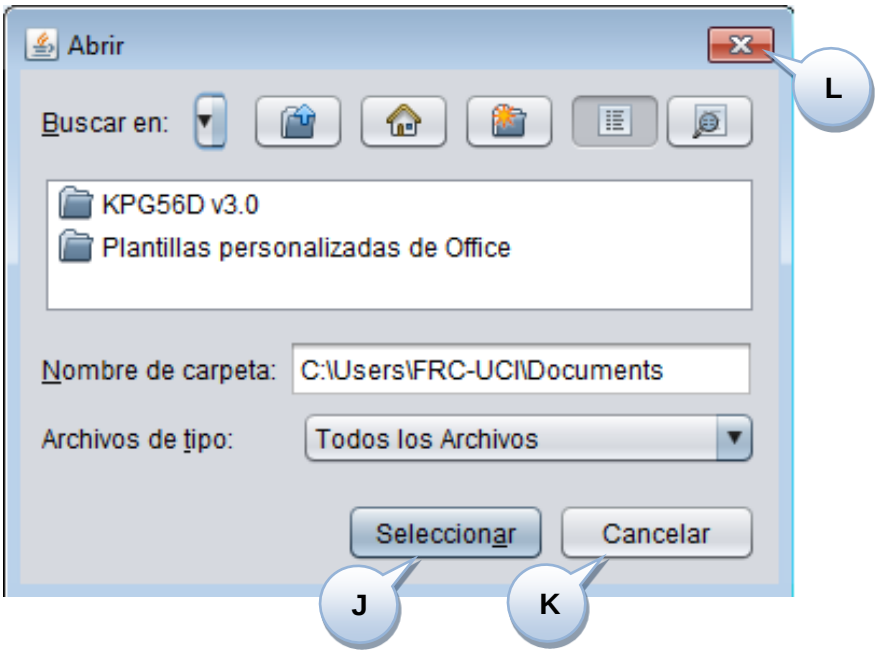


Fig. 6: Prototipo de interfaz “Selector de archivos”.

Interfaz 4

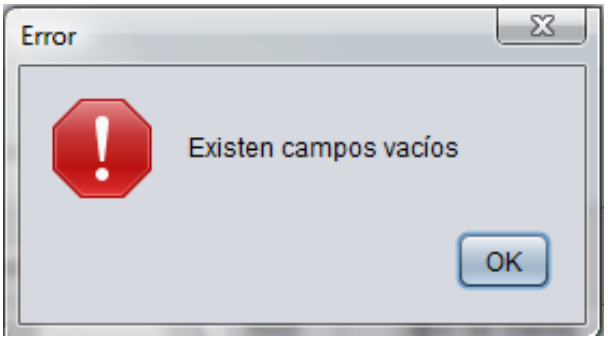


Fig. 7: Prototipo de interfaz “Mensaje de error”.

Flujos alternos	
Acción del Actor	**Respuesta del Sistema**

1. El actor Usuario selecciona la opción “Cancelar” (H).	**2.** El sistema oculta la ventana de cargar personalización.
3. El actor Usuario selecciona la opción “Cerrar” (I).	**4.** El sistema oculta la ventana de cargar personalización.
5. El actor Usuario selecciona la opción “Cancelar” (K).	**6.** El sistema oculta la ventana de selección de archivos.
7. El actor Usuario selecciona la opción “Cerrar” (L).	**8.** El sistema oculta la ventana de selección de archivos.
9. El actor Usuario selecciona la opción “Cancelar” (K).	**10.** El sistema oculta la ventana de selección de archivos.
11. El actor Usuario selecciona la opción “Cerrar” (L).	**12.** El sistema oculta la ventana de selección de archivos.
13. El actor usuario selecciona la opción “Cargar” (G).	**14.** El sistema verifica si faltan campos por llenar y muestra un mensaje de error indicando que existen campos vacíos. Ver interfaz 4.
Poscondiciones:	El sistema queda con los datos cargados de la personalización.

Tabla 5: Descripción del caso de uso “Generar paquete de instalación”.

Caso de Uso:	Generar paquete de instalación.
Actores:	Usuario
Propósito:	Este caso de uso se realiza con el objetivo de que el usuario pueda generar un instalador de una personalización de GeneSIG.

Resumen:	El caso de uso se inicia cuando el actor selecciona la opción “Generar paquete de instalación”. El caso de uso termina una vez finalizada alguna de las opciones que permite realizar.
Precondiciones:	El sistema debe estar ejecutándose correctamente. Debe existir una personalización cargada en el sistema.
Referencias:	RF 10.
Prioridad:	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor Usuario selecciona la opción “Generar paquete de instalación” (A), como se muestra en la Interfaz 1.	**2.** El sistema muestra una ventana donde se controla el proceso de creación del instalador. Ver interfaz 2.
3. El actor Usuario selecciona la opción “Generar Instalador” (B).	**4.** El sistema crea el paquete de instalación de la personalización previamente cargada, terminando así el caso de uso.

Prototipos de Interfaz

Interfaz 1

Fig. 8: Prototipo de interfaz correspondiente al menú principal de la aplicación.

Interfaz 2

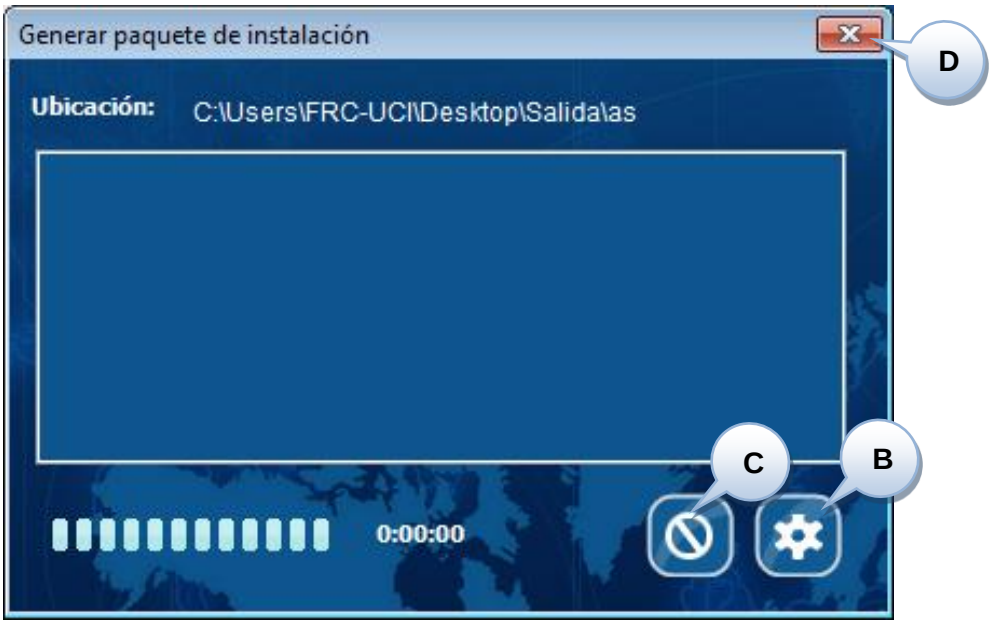


Fig. 9: Prototipo de interfaz “Generar paquete de instalación”.

Flujos alternos	
Acción del Actor	**Respuesta del Sistema**
1. El actor Usuario selecciona la opción “Cancelar” (C).	**2.** El sistema cancela la operación y oculta la ventana “Generar paquete de instalación”.
3. El actor Usuario cierra la ventana “Generación de Instaladores” (D).	**4.** El sistema cancela la operación y oculta la ventana “Generar paquete de instalación”.
5. El actor Usuario selecciona la opción “Generar Instalador” (B).	**6.** El sistema muestra un mensaje con los problemas ocurridos durante el proceso de creación del paquete de instalación.
Poscondiciones:	El sistema crea el instalador asociado a la personalización.

2.4 Descripción de los elementos de la arquitectura

Existe una gran variedad de definiciones para el término Arquitectura de Software (ASW). Una de estas es la realizada por IEEE[13] en el año 2000, donde define la ASW como el nivel conceptual más alto de un sistema en su ambiente, la organización fundamental descrita en términos de componentes, la relación entre ellos con el ambiente y los principios que guían su diseño y evolución (Sánchez Perodín, 2010).

La arquitectura de software constituye un conjunto de patrones y abstracciones para guiar la construcción de un sistema. Provee los elementos necesarios para que analistas, diseñadores y programadores trabajen en una línea común.

2.4.1 Estilos y patrones arquitectónicos

La diferencia existente entre los términos estilo arquitectónico y patrón arquitectónico no ha sido aclarada, algunos autores establecen diferencias sutiles entre ambos conceptos, pero otros lo utilizan indistintamente. De cualquier forma, los estilos y los patrones establecen un vocabulario común, y brindan soporte a los ingenieros para conseguir una solución que haya sido aplicada con éxito anteriormente, ante ciertas situaciones de diseño (Camacho, y otros, 2004). Su empleo incrementa la productividad, las probabilidades de éxito en el diseño arquitectónico y disminuye el tiempo de desarrollo.

En (Camacho, y otros, 2004) se define estilo arquitectónico como *"una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción".*

Además se exponen los principales estilos arquitectónicos:

- **Datos centralizados:** sistemas en los cuales cierto número de clientes accede y actualiza datos compartidos de un repositorio de manera frecuente.
- **Flujo de datos:** el sistema es visto como una serie de transformaciones sobre piezas sucesivas de datos de entrada. El dato ingresa en el sistema, y fluye entre los componentes, de uno en uno, hasta que se le asigne un destino final (salida o repositorio).

[13] Institute of Electrical and Electronics Engineers o Instituto de Ingenieros Eléctricos y Electrónicos.

- **Máquinas virtuales:** simulan alguna funcionalidad que no es nativa al hardware o software sobre el que está implementado.
- **Llamada y retorno:** el sistema se constituye de un programa principal que tiene el control del sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas.
- **Componentes independientes:** consiste en un número de procesos u objetos independientes que se comunican a través de mensajes.

Por otro lado están los patrones arquitectónicos, según Carlos Canal en su tesis doctoral (Canal Velasco, 2000) estos son *"esquemas de organización de los sistemas de software que determinan cuál va a ser la estructura de los mismos mediante el establecimiento de su división en subsistemas, indicando las responsabilidades de cada uno de estos subsistemas y las reglas y criterios que rigen las relaciones entre ellos".*

Entre los patrones arquitectónicos que se exponen en (Camacho, y otros, 2004) se encuentran:

- **Capas:** consiste en estructurar aplicaciones que pueden ser descompuestas en grupos de subtareas, las cuales se clasifican de acuerdo a un nivel particular de abstracción.
- **Modelo - Vista - Controlador:** divide una aplicación interactiva en tres componentes. El modelo contiene la información central y los datos. Las vistas despliegan información al usuario y el controlador captura la entrada del usuario.

Selección del estilo arquitectónico

Se selecciona el estilo arquitectónico llamada y retorno, el cual es una descomposición jerárquica en componentes que solucionan una tarea o función definida. Permite al diseñador del software la construcción de estructuras de programas fáciles de modificar y ajustar. En esta familia se encuentran, entre otros, los sistemas orientados a objetos y los sistemas jerárquicos en capas.

Selección del patrón arquitectónico

Para el desarrollo de la herramienta se hará uso del patrón arquitectónico en capas. La función fundamental de este patrón es organizar la estructura lógica de gran escala de un sistema en capas separadas de responsabilidades distintas y relacionadas, con una separación clara y cohesiva de intereses, como que las

capas "más bajas" son servicios generales de bajo nivel, y las capas más altas son más específicas de la aplicación. Además la colaboración y el acoplamiento son desde las capas más altas hacia las más bajas; se evita el acoplamiento de las capas más bajas a las más altas (Larman, 2002). Este patrón admite optimizaciones y refinamientos, además de favorecer la reutilización. Una de sus ventajas es que si existe algún error o la necesidad de realizar cambios obligatorios, solo se necesita cambiar el nivel en cuestión, sin que se afecte el correcto funcionamiento del resto del sistema.

Específicamente se empleará el patrón de tres capas, que separa el sistema en tres capas lógicas distintas. La primera se denomina capa de presentación y consiste en una interfaz gráfica de usuario, que permite la interactividad entre el usuario y el sistema. La capa intermedia consiste en la lógica del negocio y es básicamente el código al que recurre la capa de presentación para recuperar los datos deseados, es donde se establecen todas las reglas que deben cumplirse. La tercera capa es la de acceso a datos y es la encargada de almacenar los datos del sistema y de los usuarios.

2.4.2 Patrones de diseño

Los patrones del diseño tratan los problemas del diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, son soluciones exitosas a problemas comunes (Cuesta Villa, 2007). Estos patrones se encuentran separados en dos grupos: GRASP[14] y GoF[15]. Para el desarrollo de la herramienta se hace uso de patrones de diseño pertenecientes a ambos grupos, estos son:

Patrones GRASP:

- **Bajo Acoplamiento:** impulsa la asignación de responsabilidades de manera que su localización no incremente el acoplamiento, es decir, que al asignar las responsabilidades se mantenga bajo acoplamiento. Soporta el diseño de clases que son más independientes, lo que reduce el impacto del cambio.

[14] General Responsibility Assignment Software Patterns o Patrones Generales de Software para la Asignación de Responsabilidades.

[15] Gang-of-Four o Banda de Cuatro.

- **Alta Cohesión:** la cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Un elemento con responsabilidades altamente relacionadas, y que no hace una gran cantidad de trabajo, tiene alta cohesión. Ejemplo de estos elementos pueden ser las clases y subsistemas. Una clase con alta cohesión es ventajosa porque es relativamente fácil de mantener, entender y reutilizar.
- **Controlador:** no pertenece a la interfaz de usuario, es responsable de recibir o manejar un evento del sistema. Define el método para la operación del sistema. Una clase controladora diseñada incorrectamente presentará baja cohesión: estará dispersa y tendrá demasiadas áreas de responsabilidad.
- **Experto:** es un principio de guía básico que se utiliza continuamente en el diseño de objetos. El Experto asigna una responsabilidad a la clase que maneja la información necesaria para cumplir con dicha responsabilidad, solucionando así el problema de asignarla de forma general. Se mantiene el encapsulamiento de la información, puesto que los objetos utilizan su propia información para llevar a cabo las tareas. Normalmente, esto conlleva un bajo acoplamiento, lo que da lugar a sistemas más robustos y más fáciles de mantener.
- **Creador:** guía la asignación de responsabilidades relacionadas con la creación de objetos, es decir, asigna a una clase (A) la responsabilidad de crear una instancia de otra (B). Se soporta el bajo acoplamiento, lo que implica menos dependencias de mantenimiento y mayores oportunidades para reutilizar (Larman, 2002).

Patrones GoF:

- **Fachada:** oculta un subsistema detrás de un objeto, es decir, brinda una interfaz unificada para acceder a una interfaz o grupo de interfaces de un subsistema. Es el único punto de entrada para los servicios del subsistema. Esto posibilita que los usuarios no necesiten conocer las clases que hay detrás de la fachada.
- **Singleton:** garantiza que solamente se cree una instancia de la clase y provee un punto de acceso global a la misma. Todos los objetos que utilizan una instancia de esa clase usan la misma instancia. Este patrón se basa en convertir al constructor de la clase en privado evitando la creación de instancias del mismo. Para instanciar la clase el objeto se realiza a través de un método público y

estático el cual verifica si el objeto ha sido instanciado antes y devuelve la referencia. En caso de no existir lo crea y devuelve la instancia creada.

2.5 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además el modelo de diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental de las actividades de implementación (Rumbaugh, y otros, 2000).

2.5.1 Diagrama de clases del diseño

El diagrama de clases del diseño se utiliza para describir de forma gráfica las especificaciones de los objetos o clases en un sistema y las relaciones existentes entre ellos. Se considera el enlace entre los requerimientos y la implementación del mismo.

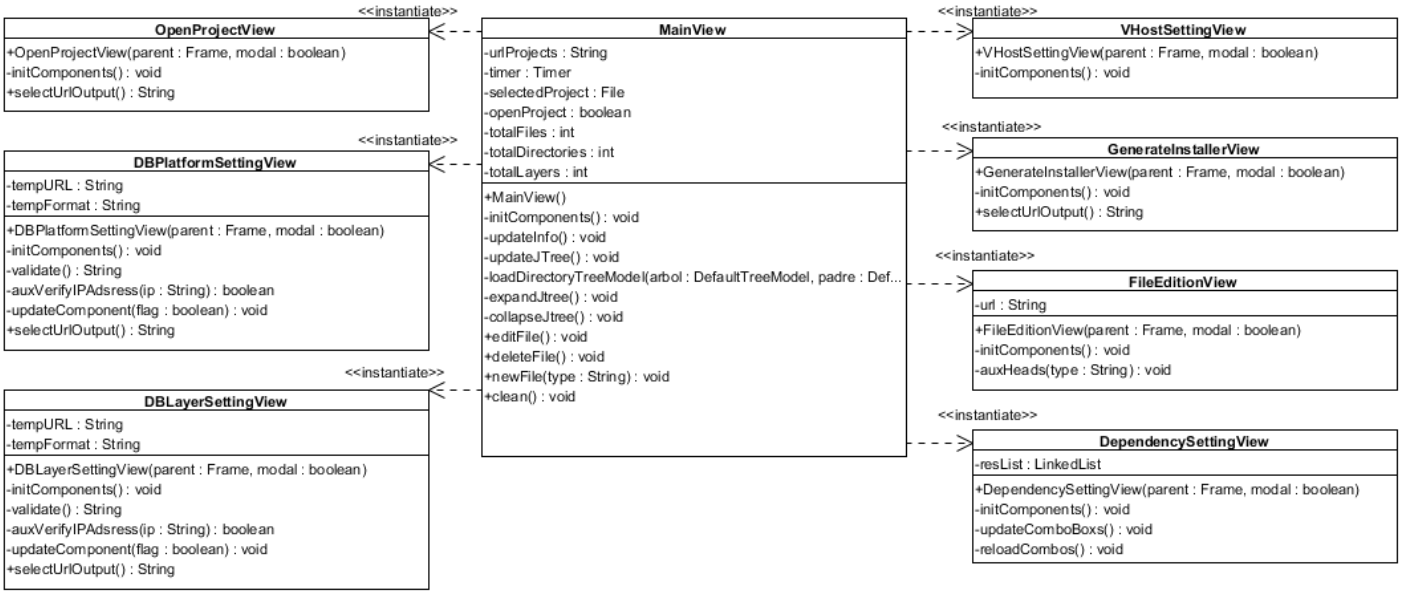


Fig. 10: Clases pertenecientes a la capa de presentación.

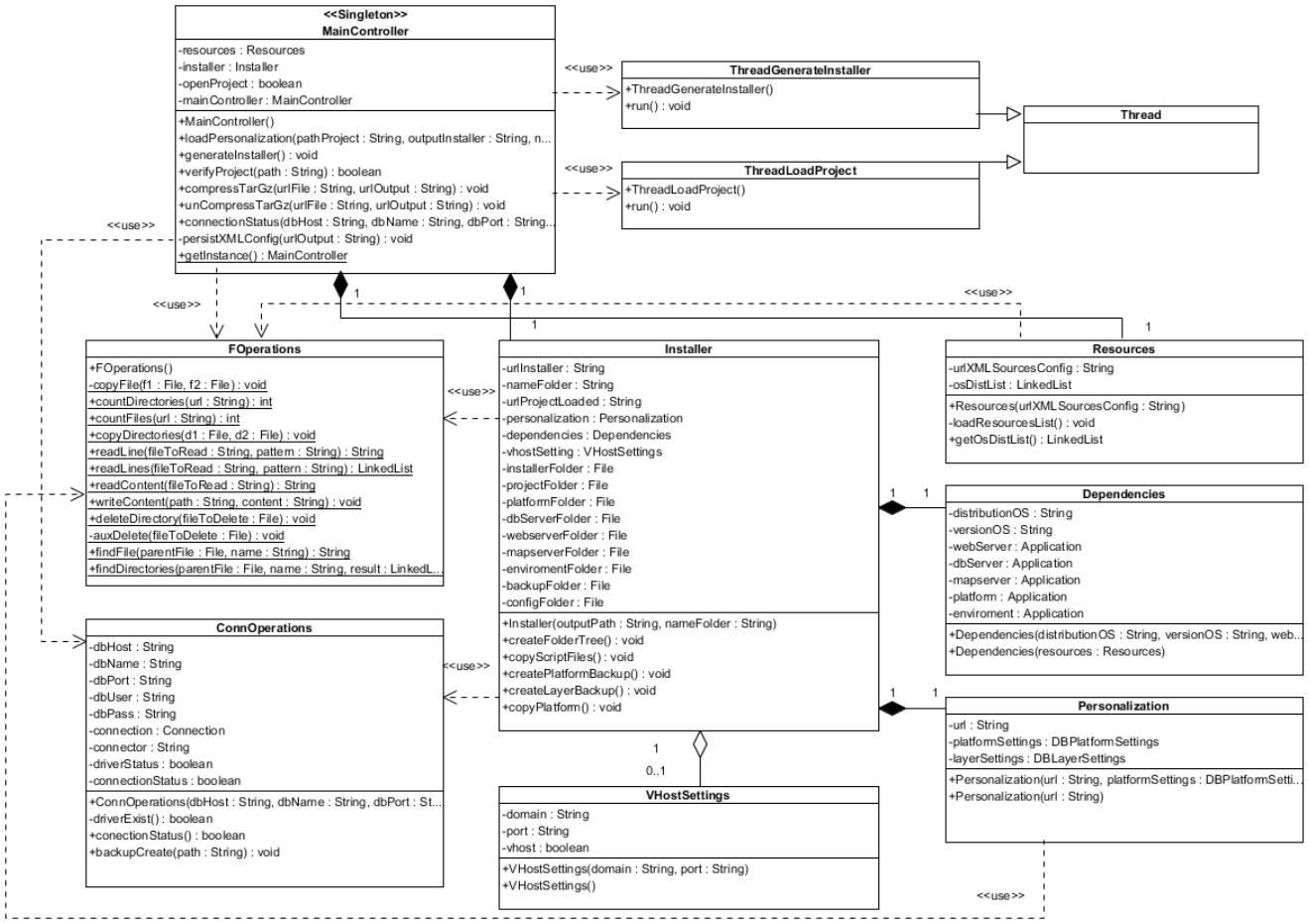


Fig. 11: Clases pertenecientes a la capa lógica del negocio.

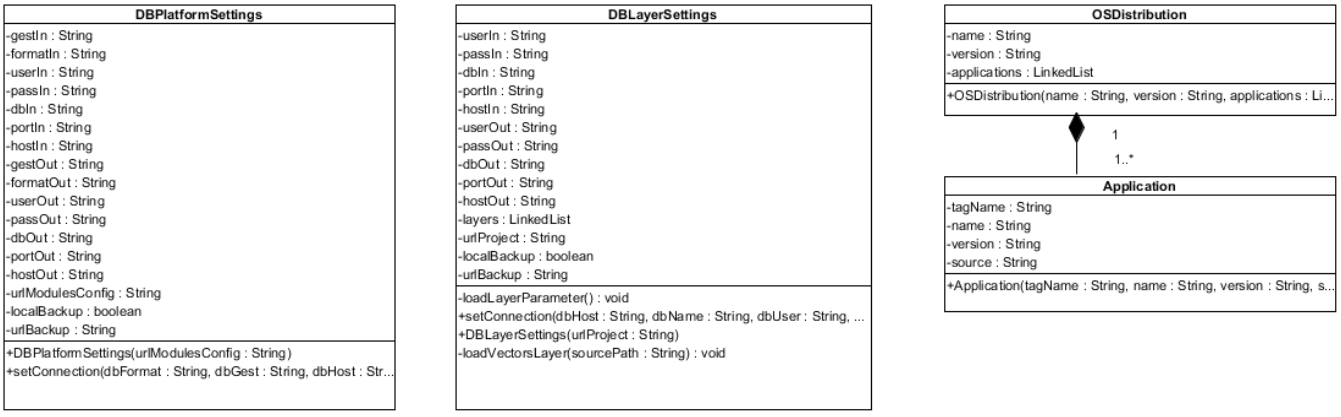


Fig. 12: Clases pertenecientes a la capa acceso a datos.

A continuación se muestra un diagrama donde se representan las relaciones existentes entre las clases de las diferentes capas. Es importante señalar que solo están presentes las clases que tienen relación con otras que pertenecen a capas diferentes. Otro aspecto a tener en cuenta es que en la capa “Presentación” no se especificó ninguna clase ya que todas cuentan con la misma relación.

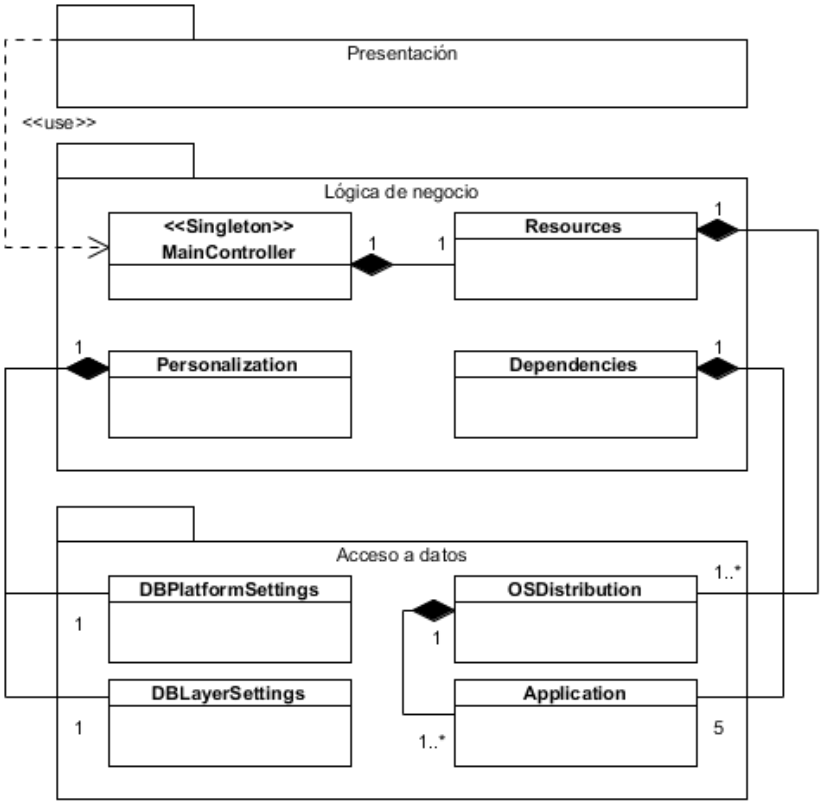


Fig. 13: Representación de las relaciones entre las clases de las diferentes capas.

2.5.2 Descripción de las clases del diseño

La descripción de las clases del diseño proporciona un mejor entendimiento de las mismas para su posterior implementación. A continuación se presenta la descripción de la clase MainController, el resto de las descripciones pueden ser consultadas en los artefactos ingenieriles de la investigación.

Tabla 6: Descripción de la clase MainController.

MainController	
Tipo de clase:	**Controladora**
Atributos	**Tipo**
resources	Resources
installer	Installer
openProject	boolean
urlBackup	String
Para cada responsabilidad:	
Nombre:	loadPersonalization(String pathProject, String outputInstaller, String nameInstaller)
Descripción:	Carga la personalización y los parámetros de configuración de la misma.
Nombre:	generateInstaller()
Descripción:	Genera el paquete de instalación correspondiente a la personalización cargada.
Nombre:	verifyProject(String path)
Descripción:	Verifica la validez de un proyecto comprobando la existencia de los ficheros de configuración.
Nombre:	compressTarGz(String urlFile, String urlOutput)
Descripción:	Comprime el paquete especificado en la ruta de salida indicada.
Nombre:	unCompressTarGz(String urlFile, String urlOutput)
Descripción:	Descomprime el paquete especificado en la ruta de salida indicada.

Nombre:	connectionStatus(String dbHost, String dbName, String dbPort , String dbUser, String dbPass)
Descripción:	Verifica el estado de conexión con la BD especificada.
Nombre:	persistXMLConfig(String urlOutput)
Descripción:	Crea un fichero XML con la configuración del paquete de instalación.
Nombre:	getInstance()
Descripción:	Devuelve la instancia de la clase, en caso de ser la primera vez que se llama crea la instancia en caso contrario devuelve la instancia ya creada.

Conclusiones del capítulo

En el presente capítulo se generaron los artefactos de la metodología de desarrollo necesarios para la implementación de la solución. La descripción de los conceptos del dominio, las relaciones existentes entre estos y el diagrama de clases del dominio correspondiente brindaron una perspectiva detallada del sistema. Se identificaron y describieron los requisitos donde se obtuvieron 10 requisitos funcionales y 5 requisitos no funcionales. Estos, fueron agrupados en casos de uso de los que fueron identificados los casos de uso arquitectónicamente significativos que son determinantes en la construcción del sistema. El empleo de patrones en el desarrollo de la herramienta posibilitó la reutilización de código y que la misma sea más fácil de comprender, mantener y extender. Se obtuvo una estructura conocida por todos los programadores lo que proporciona el ahorro de tiempo en la construcción y/o actualización de funcionalidades del software desarrollado.

CAPÍTULO 3. IMPLEMENTACIÓN Y VALIDACIÓN DE LA HERRAMIENTA PARA LA GENERACIÓN DE PAQUETES DE INSTALACIÓN

En el presente capítulo se exponen los principales elementos de la implementación del sistema, se realiza el diagrama de componentes, se abordan las estrategias de codificación y se especifican cuáles son los mecanismos para el tratamiento de errores. Se explica el modelo de despliegue y se le realizan pruebas a la solución para validar y asegurar el correcto funcionamiento de la misma.

3.1 Diagrama de componentes

Este diagrama muestra los componentes de software y las relaciones entre ellos. Especifica los elementos estructuradores de alto nivel (paquetes, librerías) de un sistema y expresa algunas de las propiedades externamente visibles de los componentes (ICESI, 2010).

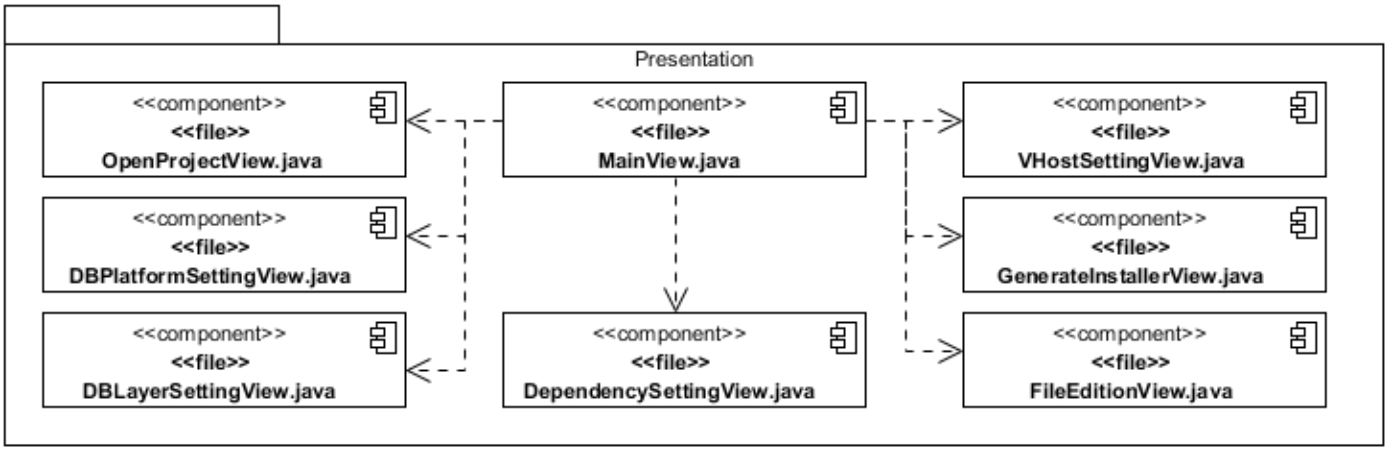


Fig. 14: Componentes pertenecientes al paquete Presentation.

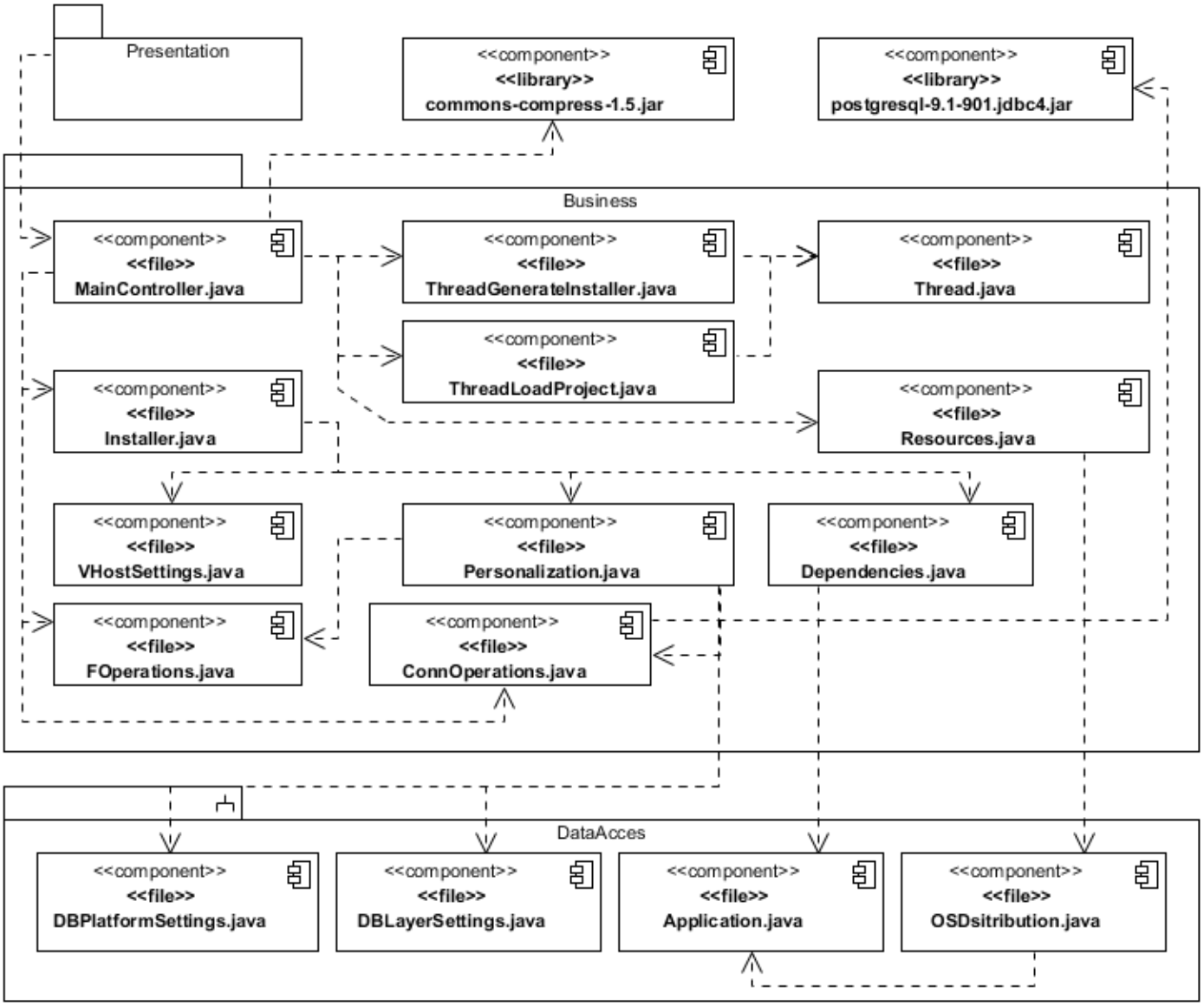


Fig. 15: Diagrama de componentes.

3.2 Tratamiento de errores

Durante la ejecución de un programa independientemente del lenguaje en que sea escrito siempre está presente la posible ocurrencia de errores que altere o interrumpa el flujo normal de las sentencias. Una de las estrategias para contrarrestar estas ocurrencias es el manejo de excepciones. Java posee un

mecanismo que permite gestionar errores y situaciones excepcionales constituyendo uno de los pilares fundamentales del lenguaje.

El empleo de excepciones para el tratamiento de errores en los programas Java destaca las ventajas que este posee sobre las técnicas de manejo de errores tradicionales. En la programación tradicional, la detección, el informe y el manejo de errores se convierten en un proceso engorroso. Por otra parte el uso de excepciones en Java permite separar el manejo de errores del código normal, posibilita la propagación de los errores sobre la pila de llamadas y permite la agrupación y diferenciación de los mismos (Lou Torrijos, 2010).

Para manejar las excepciones se emplean los bloques **try** y **catch**. Dentro del bloque **try** va el código que puede lanzar la excepción y para capturarla se utiliza **catch**, dentro de este último se escribe el código que trata el error generado.

```
30      public Resources(String urlXMLSourcesConfig) {
31          this.urlXMLSourcesConfig = urlXMLSourcesConfig;
32          osDistList = new LinkedList<>();
33          try {
34              loadResourcesList();
35          } catch (Exception e) {
36              JOptionPane.showMessageDialog(null, e.getMessage());
37          }
38      }
```

Fig. 16: Empleo de excepciones para el tratamiento de errores.

3.3 Estrategias de codificación

Los estándares de codificación son reglas que se siguen para lograr uniformidad en la escritura del código. Su empleo facilita la comprensión del sistema y propicia que diferentes programadores trabajen de forma coordinada. Para el desarrollo del sistema se utiliza el estándar de codificación Camel, el cual posee dos variantes:

- UpperCamelCase: primera letra en mayúscula, al igual que la primera letra de cada palabra interna.

- lowerCamelCase: primera letra en minúscula y la primera letra de cada palabra interna en mayúscula.

Se utiliza UpperCamelCase para el nombre de las clases y lowerCamelCase para el nombre de métodos y variables. Los comentarios deben escribirse de forma abreviada y siempre en tercera persona, evitando en todo momento el uso abusivo de los mismos. En las declaraciones no deben dejarse espacios en blanco entre el nombre de un método y el paréntesis «(» que abre su lista de parámetros. Además la llave de apertura «{» aparece al final de la misma línea de la sentencia de declaración y la de cierre «}» empieza una nueva línea luego de todas las líneas de sentencias.

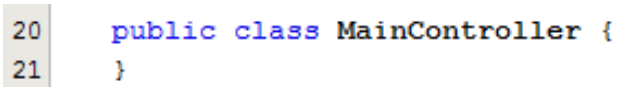


Fig. 17: Declaración de una clase empleando UpperCamelCase.

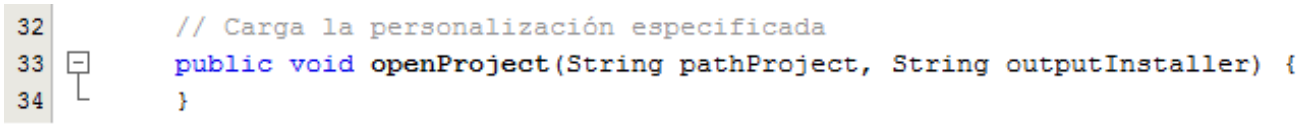


Fig. 18: Declaración de métodos y variables empleando lowerCamelCase.

3.4 Modelo de despliegue

Según (Rumbaugh, y otros, 2000) el modelo de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. Es empleado como entrada fundamental en las actividades de implementación debido a que la distribución del sistema tiene una influencia principal en su diseño.

Para el despliegue del sistema se requiere de una estación cliente la cual dentro de las características mínimas que debe poseer se encuentran el sistema operativo GNU/Linux o Windows y microprocesador Intel® Dual-Core o Intel® Core-2 Duo a 1.8 GHz. La capacidad mínima de memoria RAM debe ser de 512 MB y una capacidad de 40 GB de disco duro. Para el funcionamiento de la herramienta es imprescindible la presencia del JRE el cual es el requerimiento mínimo para poder ejecutar una aplicación Java.

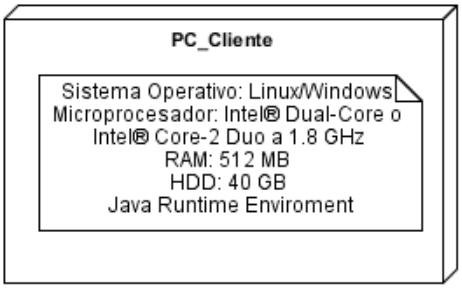


Fig. 19: Diagrama de despliegue.

3.5 Pruebas

Las pruebas de software se llevan a cabo durante todo el ciclo de vida de un proyecto y se realizan con el objetivo de descubrir errores que no han sido detectados y proporcionar una buena indicación de la fiabilidad del sistema y calidad del mismo. Los métodos de pruebas son:

- **Prueba de caja blanca o prueba de caja de cristal:** método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca se pueden obtener casos de prueba que garanticen la ejercitación por lo menos una vez de todos los caminos independientes de cada módulo y que se ejerciten las estructuras internas de datos para asegurar su validez.
- **Prueba de caja negra o prueba de comportamiento:** se centra en los requisitos funcionales del software, permitiendo obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Este método de prueba intenta encontrar errores de las siguientes categorías: funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a bases de datos externas (Pressman, 2008).

Para realizar las pruebas al sistema se empleará la técnica de pruebas de caja negra: partición equivalente. Esta técnica es una de las más efectivas, divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba a desarrollar (Pressman, 2008).

3.5.1 Caso de prueba del caso de uso Cargar personalización

A continuación se muestra el diseño de caso de prueba correspondiente al caso de uso “Cargar personalización”, para consultar el resto de los diseños de casos de prueba remitirse a los artefactos ingenieriles de la investigación.

Tabla 7: Escenarios de prueba del caso de uso “Cargar personalización”.

Nombre del caso de uso	**Descripción general**	**Escenario de prueba**	**Flujo del escenario**
Cargar personalización	Permite cargar una personalización de la plataforma GeneSIG a la cual se le va a generar un paquete de instalación.	E.P 1.1: Ingresar los datos correctamente	1. El usuario accede a la funcionalidad “cargar personalización”. 2. El sistema muestra una interfaz solicitando los parámetros necesarios para realizar la operación. 3. El usuario proporciona los datos correctamente y presiona el botón “Cargar”. 4. El sistema carga la personalización y muestra los parámetros de configuración que esta posee.
		E.P 1.2: Ingresar los datos incorrectamente.	1. El usuario accede a la funcionalidad “cargar personalización”. 2. El sistema muestra una interfaz solicitando los parámetros necesarios para realizar la operación. 3. El usuario proporciona los datos incorrectamente o deja campos

			obligatorios vacíos y presiona el botón “Cargar”. 4. El sistema muestra un mensaje de error explicando la causa del mismo. 5. El sistema brinda la posibilidad de que el usuario pueda realizar la operación nuevamente.
		E.P 1.3: Cancelar operaciones.	1. El usuario accede a la funcionalidad “cargar personalización”. 2. El sistema muestra una interfaz solicitando los parámetros necesarios para realizar la operación. 3. El usuario proporciona o no los datos y presiona el botón “Cancelar”. 4. El sistema cierra la interfaz “Cargar personalización”.

Tabla 8: Descripción de las variables del diseño de caso de prueba “Cargar personalización”

No	Nombre del campo	Clasificación	Puede ser nulo	Descripción
1	Proyecto	Campo de texto.	No	Valor de la ruta de la personalización a cargar.
2	Destino	Campo de texto.	No	Valor de la ruta de destino del paquete de instalación.
3	Nombre	Campo de texto.	No	Valor del nombre del paquete de instalación.

Tabla 9: Matriz de datos a probar en el diseño de caso de prueba “Cargar personalización”

Id del Esc.	Escenario	Var 1	Var 2	Var 3	Respuesta del sistema
E.P 1.1	Ingresar datos correctamente.	V	V	V	Satisfactoria
E.P 1.2	Ingresar datos incorrectamente.	I	V	V	Mensaje de error.
E.P 1.3	Cancelar operaciones.	I	I	V	Satisfactoria

3.5.2 Resultados de las pruebas

La aplicación de la técnica partición equivalente, correspondiente a las pruebas de caja negra se realizó con éxito mediante tres iteraciones. En la primera iteración fueron detectadas 9 no conformidades, en la segunda 5 y en la tercera no se detectaron no conformidades.

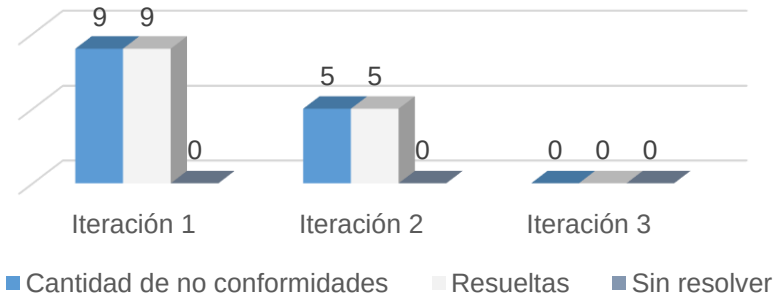


Fig. 20: Resultados de las no conformidades por iteraciones.

Mediante la utilización de un paquete de instalación se disminuye el tiempo de despliegue. Esto es debido a que el personal encargado del despliegue no debe poseer conocimientos avanzados del sistema operativo. La ubicación de los diferentes directorios en el sistema se realiza por el instalador y la edición de los ficheros de configuración se realiza de manera transparente para el usuario. Para lograr una idea más precisa se realiza un experimento que permite obtener el tiempo empleado en el despliegue de una personalización de forma manual y a través de un paquete de instalación generado con la herramienta.

Para realizar los despliegues de prueba se empleó una estación con microprocesador Intel® i3-2130 a 3.40 GHz y 2 GB de memoria RAM, además se realizó el mismo experimento en otra estación con microprocesador Intel® Core-2 Duo a 2.20 GHz y 1 GB de memoria RAM. Estos experimentos fueron realizados directamente con el cliente garantizando que los despliegues fueran desarrollados por personal altamente calificado y con experiencia en la instalación y configuración de personalizaciones de la plataforma GeneSIG. Durante la instalación y configuración de la personalización de forma manual no ocurrieron errores, las operaciones necesarias se realizaron sincronizadas y con el menor tiempo posible entre las mismas. La tabla 10 muestra los resultados obtenidos.

Tabla 10: Comparación del tiempo de despliegue realizado de forma manual y a través de un instalador.

Propiedades de la estación	**Modo de instalación**	**Tiempo (minutos)**	**Diferencia**
Intel® i3-2130 a 3.40 GHz y 2 GB de memoria RAM	Manual	20	15 (75%)
	Instalador	5	
Intel® Core-2 Duo a 2.20 GHz y 1 GB de memoria RAM	Manual	23	16 (69.5 %)
	Instalador	7	

Como se observa en la tabla anterior el empleo de un paquete de instalación generado con la herramienta puede disminuir considerablemente el tiempo requerido si el despliegue se realiza de forma manual, esto contribuye significativamente al ahorro de tiempo y esfuerzo por parte del personal.

Conclusiones del capítulo

En el presente capítulo se expusieron los artefactos de la metodología Open UP necesarios para la implementación y validación de la solución. El empleo de estándares de codificación permitió una mejor legibilidad y coherencia del código facilitando el mantenimiento del mismo. Se realizaron pruebas de caja negra, específicamente la técnica partición equivalente para verificar el cumplimiento de los requisitos

funcionales definidos para el sistema. A través de estas, se identificaron los errores y las fallas de la herramienta, lo que permitió brindarles especial atención para su posterior solución.

CONCLUSIONES

El desarrollo satisfactorio de las tareas de la investigación posibilitó dar cumplimiento al objetivo general y arribar a las siguientes conclusiones:

- El estudio de los fundamentos teóricos para la generación de paquetes de instalación permitió la selección de la metodología de desarrollo, las tecnologías y las herramientas necesarias para la elaboración del sistema.
- Mediante el análisis y diseño de la propuesta de solución se obtuvo una visión general del sistema sentando las bases para su posterior implementación.
- Como resultado de la fase de construcción se obtuvo una herramienta portable y multiplataforma desarrollada en su totalidad con herramientas Open Source.
- El empleo de los paquetes de instalación generados con la herramienta permite agilizar el proceso de instalación y configuración de personalizaciones de la plataforma GeneSIG, posibilitando además la futura edición de las configuraciones de las mismas.
- Mediante las pruebas de software realizadas se verificó el correcto funcionamiento del sistema, obteniéndose una herramienta completa en cuanto al manejo de errores.
- Los diferentes artefactos generados durante el proceso de desarrollo pueden ser empleados para la comprensión, modificación o la incorporación de nuevas funcionalidades.

RECOMENDACIONES

Concluida la presente investigación y considerando cumplidos los objetivos propuestos se recomienda:

- Enriquecer la herramienta añadiendo las dependencias, funcionalidades y configuraciones necesarias que permitan generar paquetes de instalación del resto de los productos desarrollados en la Línea de Productos de Software Aplicativos SIG.

REFERENCIAS BIBLIOGRÁFICAS

Cabrera, Lianet González y Pompa , Enrique Roberto. 2010. *Extensión de Visual Paradigm for UML para el desarrollo dirigido por modelos de aplicaciones de gestión de información.*

Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel. 2004. *Arquitectura de Software. Guía de estudio.* 2004.

Canal Velasco, Carlos. 2000. *Un Lenguaje para la Especificación y Validación de Arquitecturas de Software.* Málaga : s.n., 2000.

Cuesta Villa, Madelys. 2007. *Modelo para la ayuda a la toma de decisiones en la selección de patrones de desarrollo de software.* La Habana : s.n., 2007.

Digital River Inc. 2014. MindVision Software. *Installer Vise for Windows.* [En línea] 2014. [Citado el: 19 de Abril de 2014.] http://www.mindvision.com/winvise.asp.

García Pérez, Carlos . 2006. Adictos al Trabajo. *Instalaciones mediante IzPack.* [En línea] 26 de Septiembre de 2006. [Citado el: 3 de Febrero de 2014.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=IzPack.

Gimp. 2013. Gimp. Un programa de edición de imágenes libre y gratuito. [En línea] 2013. [Citado el: 16 de Febrero de 2014.] http://gimp.es/.

González, Daniel, Diaz Vivar, Miriam y Diaz, Boris G. 2009. *Desarrollo de un servidor de mapas utilizando software libre.* 2009. ISBN.

ICESI. 2010. Universidad ICESI. *Diagrama de componentes.* [En línea] 16 de Octubre de 2010. [Citado el: 14 de Abril de 2014.]

Inteco. 2009. *INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA.* 2009.

Langlé Campos, Rubén. 2010. Sistema de Información Geográfica. *¿Qué es un SIG?* [En línea] 2010. [Citado el: 12 de noviembre de 2013.] http://langleruben.wordpress.com/%C2%BFque-es-un-sig/.

Larman, Craig. 2002. *UML y Patrones. Introducción al análisis y diseño orientado a objetos. Segunda Edición.* s.l. : Prentice Hall, 2002.

Lasso, Iván. 2008. Proyecto Autodidacta. [En línea] 9 de Mayo de 2008. [Citado el: 19 de Enero de 2014.] http://www.proyectoautodidacta.com/comics/%C2%BFque-es-un-instalador/.

Latorre, Galo. 2010. Programación II. *JVM - JDK - JRE - Conceptos Fundamentales de la P.O.O.* [En línea] 22 de Marzo de 2010. [Citado el: 21 de Marzo de 2014.] http://gl-epn-programacion-ii.blogspot.com/2010/03/jvm-jdk-jre-conceptos-fundamentales-de.html.

Lou Torrijos, Ricard. 2010. Programación en Castellano. *Manejo de Errores Usando Excepciones Java.* [En línea] 2010. [Citado el: 17 de Abril de 2014.] http://www.programacion.com/articulo/manejo_de_errores_usando_excepciones_java_98/2.

Matos García, Rosa María. 1999. *Diseño de bases de datos.* 1999.

Membrides Espinosa, Antonio, y otros. 2012. *Enfoque práctico. Desarrollo de Sistemas de Información Geográficos sobre una plataforma soberana.* La Habana : s.n., 2012.

Mep Producciones. 2007. Softpedia. [En línea] 5 de Noviembre de 2007. http://www.softpedia.es/programa-Mep-Installer-87386.html.

Mesa Rodríguez, Ofreidis . 2011. IzPack. [En línea] 11 de Octubre de 2011. [Citado el: 27 de Febrero de 2014.] http://www.ecured.cu/index.php/IzPack.

Netbeans Team. 2011. Netbeans. [En línea] 2011. [Citado el: 25 de Febrero de 2014.] https://netbeans.org/index_es.html.

Olguin Juarez, Efrain Alberto. 2013. OpenUP. *Metodología Open UP.* [En línea] 5 de Septiembre de 2013. [Citado el: 8 de Febrero de 2014.] http://openupeaojmp.blogspot.com/2013/09/metodologia-open-up.html.

Pantoja Zaldívar, Yoenis y Hernández Montero, Lidisy. 2010. *Documento Visión v1.0.* Habana : s.n., 2010.

Pressman, Roger S. 2008. *Ingeniería de Software. Un enfoque práctico.* Quinta. s.l. : Mc Graw Hill, 2008.

Roca Chillida, José Miguel. 2010. ¿Qué son las TIC? *InformeTICfacil.com.* [En línea] 2010. [Citado el: 12 de noviembre de 2013.] http://www.informeticplus.com/que-son-las-tic.

Rodríguez, José Ignacio, y otros. 2000. *Aprenda Java como si estuviera en primero.* San Sebastián : s.n., 2000.

Rumbaugh, James, Jacobson, Ivar y Booch, Grady. 2000. *El Lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : Addison Wesley, 2000.

—. 2000. *El Proceso Unificado de Desarrollo de software.* Madrid : Addison Wesley, 2000. 84-7829-036-2.

Saiz, Esteban. 2010. Genbeta. *Compila tus programas en paquetes .deb con Ubucompilator.* [En línea] 17 de Enero de 2010. [Citado el: 22 de Enero de 2014.] http://www.genbeta.com/linux/compila-tus-programas-en-paquetes-deb-con-ubucompilator.

Sánchez Perodín, Yusdenis. 2010. *Modelo para el diseño y evaluación de la arquitectura de los sistemas de gestión de información biomédica.* Ciudad de La Habana : s.n., 2010.

Sommerville, Ian. 2005. *Ingeniería de Software.* Séptima. Madrid : s.n., 2005.

UCI. 2012. Misión. [En línea] 2012. [Citado el: 12 de noviembre de 2013.] http://www.uci.cu/?q=mision.

UptoDown. 2013. Pequeño y ligero entorno de desarrollo. [En línea] 2013. [Citado el: 13 de Febrero de 2014.] http://geany.uptodown.com/.

Vega Silva, José Rafael . 2011. [En línea] 2011. [Citado el: 26 de Febrero de 2014.] http://www.ecured.cu/index.php/Servidor_web.

W3C. 2010. World Wide Web Consortium. *Guías Breves de Tecnologías W3C.* [En línea] 2010. [Citado el: 23 de Marzo de 2014.] http://www.w3c.es/Divulgacion/GuiasBreves/TecnologiasXML.

BIBLIOGRAFÍA CONSULTADA

Buschmann, Frank y Henney, Kevlin. 2008. *Pattern-Oriented Software Architecture.* Munich : s.n., 2008.

Cabrera, Lianet González y Pompa , Enrique Roberto. 2010. *Extensión de Visual Paradigm for UML para el desarrollo dirigido por modelos de aplicaciones de gestión de información.*

Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel. 2004. *Arquitectura de Software. Guía de estudio.* 2004.

Canal Velasco, Carlos. 2000. *Un Lenguaje para la Especificación y Validación de Arquitecturas de Software.* Málaga : s.n., 2000.

Cuesta Villa, Madelys. 2007. *Modelo para la ayuda a la toma de decisiones en la selección de patrones de desarrollo de software.* La Habana : s.n., 2007.

Digital River Inc. 2014. MindVision Software. *Installer Vise for Windows.* [En línea] 2014. [Citado el: 19 de Abril de 2014.] http://www.mindvision.com/winvise.asp.

Eclipse Foundation. 2012. OpenUP. *Introduction to OpenUP.* [En línea] 1 de Junio de 2012. [Citado el: 12 de Enero de 2014.] http://epf.eclipse.org/wikis/openup/index.htm.

García Pérez, Carlos . 2006. Adictos al Trabajo. *Instalaciones mediante IzPack.* [En línea] 26 de Septiembre de 2006. [Citado el: 3 de Febrero de 2014.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=IzPack.

Gimp. 2013. Gimp. Un programa de edición de imágenes libre y gratuito. [En línea] 2013. [Citado el: 16 de Febrero de 2014.] http://gimp.es/.

González, Daniel, Diaz Vivar, Miriam y Diaz, Boris G. 2009. *Desarrollo de un servidor de mapas utilizando software libre.* 2009. ISBN.

ICESI. 2010. Universidad ICESI. *Diagrama de componentes.* [En línea] 16 de Octubre de 2010. [Citado el: 14 de Abril de 2014.]

IEEE Computer Society Order. 2004. *SWEBOK.* California : s.n., 2004. ISBN: 0-7695-2330-7.

Inteco. 2009. *INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA.* 2009.

Langlé Campos, Rubén. 2010. Sistema de Información Geográfica. *¿Qué es un SIG?* [En línea] 2010. [Citado el: 12 de noviembre de 2013.] http://langleruben.wordpress.com/%C2%BFque-es-un-sig/.

Larman, Craig. 2002. *UML y Patrones. Introducción al análisis y diseño orientado a objetos. Segunda Edición.* s.l. : Prentice Hall, 2002.

Lasso, Iván. 2008. Proyecto Autodidacta. [En línea] 9 de Mayo de 2008. [Citado el: 19 de Enero de 2014.] http://www.proyectoautodidacta.com/comics/%C2%BFque-es-un-instalador/.

Latorre, Galo. 2010. Programación II. *JVM - JDK - JRE - Conceptos Fundamentales de la P.O.O.* [En línea] 22 de Marzo de 2010. [Citado el: 21 de Marzo de 2014.] http://gl-epn-programacion-ii.blogspot.com/2010/03/jvm-jdk-jre-conceptos-fundamentales-de.html.

Lou Torrijos, Ricard. 2010. Programación en Castellano. *Manejo de Errores Usando Excepciones Java.* [En línea] 2010. [Citado el: 17 de Abril de 2014.] http://www.programacion.com/articulo/manejo_de_errores_usando_excepciones_java_98/2.

Matos García, Rosa María. 1999. *Diseño de bases de datos.* 1999.

Membrides Espinosa, Antonio, y otros. 2012. *Enfoque práctico. Desarrollo de Sistemas de Información Geográficos sobre una plataforma soberana.* La Habana : s.n., 2012.

Mep Producciones. 2007. Softpedia. [En línea] 5 de Noviembre de 2007. http://www.softpedia.es/programa-Mep-Installer-87386.html.

Mesa Rodríguez, Ofreidis . 2011. IzPack. [En línea] 11 de Octubre de 2011. [Citado el: 27 de Febrero de 2014.] http://www.ecured.cu/index.php/IzPack.

Netbeans Team. 2011. Netbeans. [En línea] 2011. [Citado el: 25 de Febrero de 2014.] https://netbeans.org/index_es.html.

Olguin Juarez, Efrain Alberto. 2013. OpenUP. *Metodología Open UP.* [En línea] 5 de Septiembre de 2013. [Citado el: 8 de Febrero de 2014.] http://openupeaojmp.blogspot.com/2013/09/metodologia-open-up.html.

Pantoja Zaldívar, Yoenis y Hernández Montero, Lidisy. 2010. *Documento Visión v1.0.* Habana : s.n., 2010.

Pressman, Roger S. 2008. *Ingeniería de Software. Un enfoque práctico.* Quinta. s.l. : Mc Graw Hill, 2008.

Pressman, Roger S. 2010. *Software Engineering: A Practitioner´s Approach.* Seventh Edition. 2010. ISBN: 978-0-07-337597-7.

Roca Chillida, José Miguel. 2010. ¿Qué son las TIC? *InformeTICfacil.com.* [En línea] 2010. [Citado el: 12 de noviembre de 2013.] http://www.informeticplus.com/que-son-las-tic.

Rodríguez, José Ignacio, y otros. 2000. *Aprenda Java como si estuviera en primero.* San Sebastián : s.n., 2000.

Rumbaugh, James, Jacobson, Ivar y Booch, Grady. 2000. *El Lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : Addison Wesley, 2000.

—. 2000. *El Proceso Unificado de Desarrollo de software.* Madrid : Addison Wesley, 2000. 84-7829-036-2.

Saiz, Esteban. 2010. Genbeta. *Compila tus programas en paquetes .deb con Ubucompilator.* [En línea] 17 de Enero de 2010. [Citado el: 22 de Enero de 2014.] http://www.genbeta.com/linux/compila-tus-programas-en-paquetes-deb-con-ubucompilator.

Sánchez Perodín, Yusdenis. 2010. *Modelo para el diseño y evaluación de la arquitectura de los sistemas de gestión de información biomédica.* Ciudad de La Habana : s.n., 2010.

Sommerville, Ian. 2005. *Ingeniería de Software.* Séptima. Madrid : s.n., 2005.

UCI. 2012. Misión. [En línea] 2012. [Citado el: 12 de noviembre de 2013.] http://www.uci.cu/?q=mision.

UptoDown. 2013. Pequeño y ligero entorno de desarrollo. [En línea] 2013. [Citado el: 13 de Febrero de 2014.] http://geany.uptodown.com/.

Vega Silva, José Rafael . 2011. [En línea] 2011. [Citado el: 26 de Febrero de 2014.] http://www.ecured.cu/index.php/Servidor_web.

W3C. 2010. World Wide Web Consortium. *Guías Breves de Tecnologías W3C.* [En línea] 2010. [Citado el: 23 de Marzo de 2014.] http://www.w3c.es/Divulgacion/GuiasBreves/TecnologiasXML.

ANEXOS

Anexo 1. Preguntas realizadas en las entrevistas con el cliente para la obtención de requisitos del sistema

1. ¿Cómo se realiza el despliegue de las personalizaciones de la plataforma GeneSIG?
2. ¿Quién es el encargado de realizar este proceso?
3. ¿Cuáles son las distribuciones de GNU/Linux en las que se realiza el despliegue de las personalizaciones?
4. ¿Cuáles son las configuraciones realizadas durante el proceso de despliegue?
5. ¿Cuáles son las dependencias necesarias para la plataforma GeneSIG y su compatibilidad en las distribuciones de GNU/Linux en que se va a desplegar?
6. ¿Qué características de hardware deben poseer las estaciones donde se realiza el despliegue para lograr un funcionamiento óptimo de la personalización?
7. ¿Cuál es la información técnica necesaria para lograr un mejor entendimiento por parte del personal encargado?

Anexo 2. Prototipos de interfaces de usuario

Fig. 21: Prototipo de interfaz de usuario correspondiente a la interfaz principal.

Fig. 22: Prototipo de interfaz de usuario “Ajustes de dependencias”.

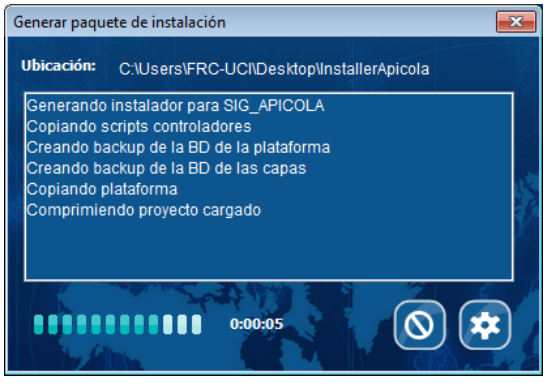


Fig. 23: Prototipo de interfaz de usuario “Generar paquete de instalación”.

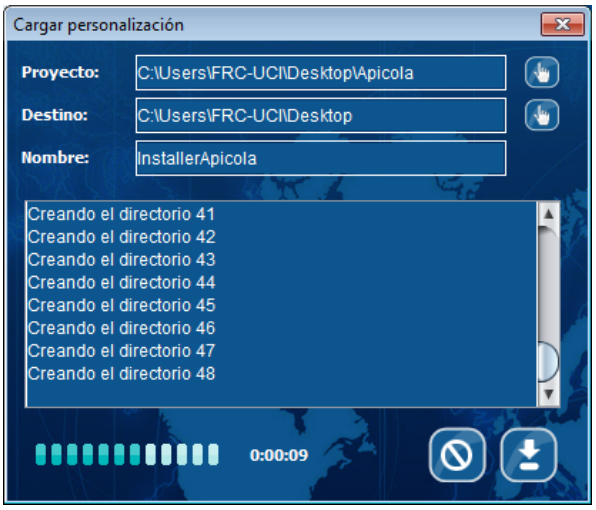


Fig. 24: Prototipo de interfaz de usuario “Cargar personalización”.

Fig. 25: Prototipo de interfaz de usuario “Ajustes de conexión de las capas”.

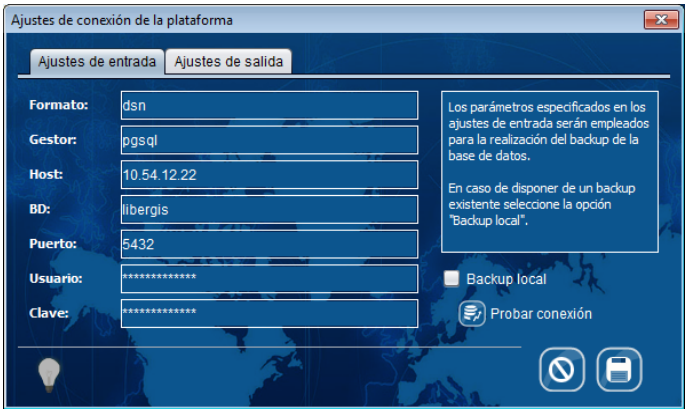


Fig. 26: Prototipo de interfaz de usuario “Ajustes de conexión de la plataforma”

GLOSARIO DE TÉRMINOS

API: representa una interfaz de comunicación entre componentes de software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos.

Backup: se utiliza para denominar las salvas que se le realizan a sistemas operativos, base de datos o grupos de informaciones con una importancia relevante.

Cartografía: ciencia que se encarga del trazado y el estudio de mapas geográficos.

Herramientas CASE: herramientas que brindan asistencia a los analistas, ingenieros de software y desarrolladores, durante el ciclo de vida de un proyecto.

Multiplataforma: término usado para referirse a los programas, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas.

Modelo raster: es cualquier tipo de imagen digital representada en mallas, se centra en las propiedades del espacio más que en la precisión de la localización. Divide el espacio en celdas regulares donde cada una de ellas representa un único valor.

Modelo vectorial: la representación de los datos vectoriales se centra en la precisión de localización de los elementos geográficos sobre el espacio. Está compuesto por líneas, puntos y polígonos.

Mapa: es una representación gráfica y métrica de una porción de territorio de acuerdo a los principios y métodos de la cartografía.

Tags: es una marca o etiqueta que delimita una región en los lenguajes basados en XML.